SIMONE CATTANI

Time Agitation Heuristic
A new constructive heuristic for the VRPTW

São Paulo
2015

SIMONE CATTANI

Time Agitation Heuristic
A new constructive heuristic for the VRPTW

Trabalho de conclusão de Curso, apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

São Paulo
2015

SIMONE CATTANI

Time Agitation Heuristic
A new constructive heuristic for the VRPTW

Trabalho de conclusão de Curso, apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do título de Bacharel em Engenharia.

Area de Concentração:

Engenharia Elètrica com ênfase
em Engenharia da Computação

Orientador:

Prof. Dr. Reginaldo Arakaki

São Paulo
2015

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do prientador: _____

Catalogação-na-publicação

Cattani, Simone
Time Agitation Heuristic, a new constructive heuristic for the VRPTW / S. Cattani -- São Paulo, 2015.
80 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Veículos 2.Heurística 3.População I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Acknowledgment

I would like to thank my supervisor prof. Reginaldo Arakaki, for the great opportunity given to me to work in a so challenging and instructive environment, allowing me to make an important experience that has certainly contributed to my professional training. Another thank goes to Marcelo Pita and Leandro Rodrigues for the constant and careful orientation during the whole project; working with you was really stimulating for me.

Desidero ringraziare tutti coloro che hanno collaborato alla realizzazione di questa tesi e che mi hanno supportato nel corso di tutta la carriera accademica.

Innanzi tutto un affettuoso grazie alla mia sorellina e ai miei genitori per il loro costante sostegno, in particolare durante questa lunga e impegnativa esperienza di doppia laurea, questa tesi è un'altra importante tappa del lungo cammino verso il quale voi mi avete indirizzato e che mi auguro vi renderà sempre orgogliosi di me. Desidero ringraziare tutta la mia famiglia, in particolare nonno Giuseppe, nonna Anna, zia Carla e la mia cuginetta Sara.

Ringrazio tutti i miei amici, con cui ho condiviso momenti di crescita professionale e soprattuto personale. Un abbraccio ai miei coscritti Samuel, Rossella, Stefano, Andrea e Sabrina, che porto sempre vicino anche quando molto lontani, e ai miei compagni. A Bert e a tutti i "Lizzardi": grazie di cuore per i bellissimi anni passati assieme a sudare sui libri condividendo gioie e dolori.

Un *obrigado* anche ad Alfonso e Filippo, con i quali ho condiviso ogni momento di quest'avventura oltre oceano, e un *obrigado* a Noelia, Poncho, Fabio, Martina, Alberto, Federica, Bruno e tutti i compagni di viaggio con i quali ho vissuto emozioni indimenticabili che mi resteranno per sempre nel cuore.

Il mio ultimo pensiero va a nonno Alberto, che per me ha sempre rappresentato un esempio e una fonte di ispirazione. Grazie di tutto nonno.

Abstract

The Vehicle Routing Problem (VRP) is one of the most important, and studied, combinatorial optimization problem, representing the core of decision support system for real-life distribution applications. Basically, a typical vehicle routing problem can be described as the computation of the optimal set of routes to be performed by a fleet of vehicles to serve a give set of customers, distributed in a certain area.

The Vehicle Routing Problem with Time Windows ($VRPTW$) is an extension of the basic model, where time constraints are introduced to define a limited time range within the service at a specific customer should start. Time windows arise naturally in problems faced by business organizations that work on fixed time schedules.

The present work aims to introduce a new $VRPTW$ constructive heuristic for the population initialization in a genetic algorithm. Our intent is designing a time constraints-driven methodology exploiting the randomic approach to focus on the resolution of the limitations introduced by the time windows. The developed algorithm is able to generate a sub-optimal solution within a really short time, reaching a quality comparable with the wide used stochastic $PFIH$ and providing interesting results in terms of diversity control.

Keywords: *vehicle routing problem, time window constraints, constructive heuristic, population generator, diversity control.*

Sommario

Il Vehicle Routing Problem (VRP) è un problema di ottimizzazione combinatoria nell'ambito della ricerca operativa. A causa della sua rilevanza all'interno dei sistemi di supporto alla decisione per le applicazioni di distribuzione di beni, il VRP rappresenta uno dei problemi maggiormente studiati e analizzati nel settore logistico. Fondamentalmente, risolvere un VRP significa determinare il miglior piano di gestione per una determinata flotta di veicoli, al fine di servire un insieme di clienti distribuiti su di un'area geografica.

Il Vehicle Routing Problem with Time Windows ($VRPTW$) è un'estensione del modello basico, nel quale vengono introdotti, per ogni cliente, dei vincoli temporali all'interno dei quali è costretto l'arrivo del veicolo e l'inizio del servizio. Le cosiddette "finestre di tempo" sono un vincolo naturale all'interno di applicazioni offerenti servizi pianificati, per questa ragione, tale variante del VRP è stata oggetto nel corso degli ultimi 20 anni di una particolare attenzione.

Il nostro lavoro ha come obiettivo introdurre una nuova euristica costruttiva per il $VRPTW$, al fine di utilizzarla all'interno di un algoritmo genetico per la generazione della popolazione iniziale. Il nostro intento è progettare una metodologia orientata ai vincoli temporali, sfruttando il concetto di "computazione non deterministica", al fine di concentrarsi sulla risoluzione delle limitazioni introdotte dalle finestre temporali. L'algoritmo sviluppato è in grado di generare soluzioni sub-ottime in un tempo realmente molto breve, raggiungendo una qualità paragonabile a quella della largamente diffusa versione stocastica del $PFIH$ e fornendo interessanti risultati per quanto riguarda il controllo della diversità.

Parole chiave: *vehicle routing problem, vincoli temporali, euristica costruttiva, generazione della popolazione, controllo della diversità*

Resumo

O Vehicle Routing Problem (VRP) é um problema de otimização combinatória no âmbito da pesquisa operacional. Por causa da relevância dele em sistemas de suporte a decisões das aplicações de distribuição dos bens, o VRP representa um dos mais estudados e analisados problemas no setor logístico. Basicamente, resolver um VRP significa determinar o melhor plano de gestão duma determinada frota de veículos, ao fim de servir um conjunto de clientes distribuídos numa área geográfica.

O Vehicle Routing Problem with Time Windows ($VRPTW$) é uma extensão do modelo básico, no qual são introduzidos, por cada cliente, vínculos de tempo entre os quais é obrigatória a chegada do veículo e o começo do serviço. As janelas de tempo são um vínculo natural nas aplicações que oferecem serviços planejados e, por esta razão, esta variante do VRP recebeu, nos últimos 20 anos, uma particular atenção.

O nosso trabalho tem como objetivo introduzir uma nova heurística construtiva pelo $VRPTW$, ao fim de utilizá-la num algoritmo genético pela geração da população inicial. O nosso intento é projetar uma metodologia orientada aos vínculos temporais, explorando o conceito de "computação não determinista" ao fim de concentrar-se na resolução das limitações introduzidas pela janelas de tempo. O algoritmo desenvolvido gera soluções sub-óptimas num tempo muito breve, chegando numa qualidade comparável com aquela da difundida versão estocástica do $PFIH$ e fornecendo interessantes resultados em termos de controle da diversidade.

Palavras chave: *problema de roteamento de veículos, vínculos com janela de tempo, heurística construtiva, gerador de população, controle da diversidade.*

List of Figures

List of Tables

Contents

Chapter 1

Introduction

In a global reality, like the one in which we live, economic opportunities are increasingly related to the mobility of people, goods and information. The relation between the quantity and the quality of transportation infrastructure and the level of economic development is clear.

When transport systems are efficient, they provide economic and social opportunities and benefits that result in positive multipliers effects such a better accessibility to markets, employment and additional investments, increasing the quality of life.

Because of its intensive use of infrastructures, the transport sector is an important component of a national economy. As reported by the United States Department of Transportation[1], in 2002 transportation-related foods and services accounted for more than 10%, over one trillion dollars, of U.S. Gross Domestic Product, and only three sectors (housing, health care and food) contributed a larger share of GDP than transportation.

The constant high growth of this sector and its important implications on both economical and environmental aspects have brought in the last years to a deep and wide research on transports optimization and operational costs minimization.

In that context, the Vehicle Routing Problem (VRP) represents one of the most important, and studied, combinatorial optimization problem. Basically, a typical vehicle routing problem can be described as the computation of the optimal set of routes to be performed by a fleet of vehicles to serve a give set of

[1] http://www.rita.dot.gov/bts/programs/freight_transportation/html/transportation.html

customers, distributed in a specific area.

In 1959, Dantzig and Ramser[12] described a real world application concerning the delivery of gasoline to service stations and proposed the first mathematical programming formulation and algorithmic approach. This seminal paper has given rise to hundreds of models and algorithm proposed to compute optimal and approximate solutions of a large variety of *VRP* variants.

The Vehicle Routing Problem with Time Windows (*VRPTW*) is one of the most known extensions of the basic model, where time constraints are introduced to define a limited time range within the service at a specific customer should be start. Time windows arise naturally in problems faced by business organizations that work on fixed time schedules. Specific examples include bank deliveries, postal deliveries, industrial refuse collection, national franchise restaurant services, school bus routing, security patrol services and just in time manufacturing.

The *VRPTW* has been subject of intensive research efforts for both heuristic and exact optimization approaches, although, dealing with modern large-sized problems, the latter has demonstrated to be an unfeasible technique. The current state-of-art heuristics consist of evolution strategies, large neighborhood searches, iterated local searches and multi-start local searches.

As reported by Toth and Vigo in their recent book[45], the methods that lately seem to produce the best results are evolutionary strategies like memetic algorithms, a powerful meta-heuristic that tries to combine different valid solutions with the intention to preserve the best partial characteristics already discovered and generate better offsprings, converging to a sub-optimal solution.

The present work aims to introduce a new constructive heuristic for the *VRPTW*. This class of algorithms, in which are classified heuristics like the *PFIH*, proposed by Solomon in 1987, generate a solution starting from the problem definition, proceeding choosing which customer has to be served by a certain vehicle and in which order, only applying decisions based on the problem characteristics.

The work is part of a wider research that aims to develop a new genetic algorithm to compete with the current enterprise solutions. Our specific focus is the formulation of a new generator that will be used to initialize the population pool, this fact implies some important consideration about performance and diversity control that will be presented in the next chapters.

Our target is the development of an algorithm able to generate a sub-optimal

solution within a really short time exploiting the randomic approach. Moreover, our intent is design a time constraints-driven methodology focused on the resolution of the limitations introduced by the time windows.

The rest of the present document will be structured as follows. The chapter 2 will provide a complete definition of the used model, formalizing the VRP and $VRPTW$ problems, furthermore will be introduced the theoretical background necessary to better understand the thesis. Then chapter 3 provides a complete panoramic of the state-of-art solutions showing the most relevant works. Chapter 4 will outline in details our solution, presenting the proposed algorithm and its behavior. Chapter 5 is dedicated to results of experimental analysis performed on available benchmark sets. Finally chapter 6 will present the conclusions coming from the tests and chapter 7 some directions for future development.

Chapter 2

Problem Formulation

The following chapter is intended to formally present the faced problem and to illustrate the theoretical foundations for a clear comprehension of the topics covered in this thesis.

The first section introduces the *VRP* problem, presenting the formal definition proposed by literature and the relative integer programming model. For a better understanding of the *VRP* family, will be firstly presented the classical *CVRP* version and then introduced the *VRPTW* variant. The last part of the first section also provides an introduction to the existing benchmark instances used to evaluate the heuristic algorithms.

Section two gives a brief introduction to computational complexity followed by a presentation of the randomized programming and the benefits derived by the application of the randomized approach to NP-complete problems.

The last section provides an overview on genetic algorithms in order to better understand the possible usage of the proposed solution in that kind of complex solver. The last part of the fourth section presents some considerations about the population diversity control in genetic algorithms and some metrics used to quantify it.

2.1 Vehicle Routing Problem

In this section it will be presented in a formal way the family of vehicle routing problems. A generic definition of this well studied issue could be: given a set of transportation requests and a fleet of vehicles, the task is

> [...] *determine a set of vehicle routes to perform all (or some) transportation requests with the given vehicle fleet at minimum cost; in particular, decide which vehicle handles which requests in which sequence so that all vehicle routes can be feasibly executed*[45].

As already reported in the previous chapter1, this kind of problem affects many distribution and transportation systems, including such examples as bank deliveries, postal deliveries, school bus routing and security patrol services. For this reason they have received considerable attention in the past decades.

The *VRP* was formally introduced for the first time in 1959 by Dantzig and Ramser[12], which introduced the *truck dispatching problem* as a real-world application in order to optimize the delivery of gasoline to gas stations. Since that moment, and until today, a huge number of researches and publications was made trying to solve new versions of *VRP* or to improve existing algorithms.

The wide interest for this research area has bring to the creation of communities composed from both academic and industrial people like the *Transportation Science and Logistic Society* (TLS) within INFORMS (Institute for Operations Research and Management Science), or the *VeRoLog* (Vehicle Routing and Logistics optimization).

During the past decades, starting from the general definition, a set of problems was introduced in order to deal with specific generalizations defining new constraints to respond to different needs coming from real world applications.

The present work was developed on the *VRPTW*, one of the most analyzed variant that introduces time constraints to the original problem. In the next paragraphs will be firstly introduced the simple *CVRP* version to better understand the basic underling problem, then the *VRPTW* variant will be presented, introducing the formalism adopted in the present document. Finally, the formal mathematical model is reported.

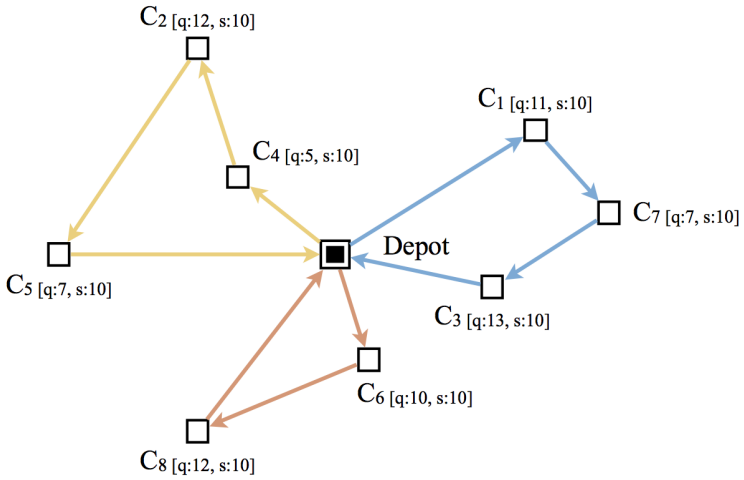


Figure 2.1: *CVRP* representation. *Source: author*

2.1.1 Capacitated Vehicle Routing Problem

As already mentioned, the *Capacitated Vehicle Routing Problem* (CVRP) represents the most relevant version of the *VRP* family. Notions and models of this basic variant is fundamental in order to better understand the proposed extensions and clarify the differences between them.

The *CVRP* request consists of the goods deliveries from a single depot to a set of customers, $N = \{1, 2, ..., n\}$. Each customer is described by its position, a demand $q_i \geq 0$, that is the amount that has to be delivered, and a service time s_i, the interval required to conclude the customer attendance. Each customer has to be served by one and only one vehicle.

The depot is usually denoted as *point 0* and doesn't have vehicles and goods quantity limitations. The fleet is composed by identical vehicles with a certain capacity Q and all of them start from and have to return to the unique depot.

The vehicle travel cost moving from i to j is denoted by c_{ij}, and usually it's assumed as travel cost the euclidean distance between the two points. This assumption is absolutely arbitrary and adopted for benchmarking and academic analysis, but for real applications can be easily replaced by geodesic distances or could be obtained from a navigation service.

Using the defined informations, the problem could be structured using a graph $G = (V, E)$, where $V = \{0\} \cup N$ and the depot is treated like a simple

node with $q_0 = 0$ and $s_0 = 0$, the edges are weighted using the chosen travel cost. If $c_{ij} = c_{ji}$, G will be an undirected graph, but in case of asymmetric costs, a complete digraph could be used, too.

A route (or tour) is defined as a sequence $r = (i_0, i_1, ..., i_s, i_{s+1})$ of the visited customers and where the first and the last customer are both a depot representation ($i_0 = i_{s+1} = 0$). Obviously the cost of the route is calculated like the summation of all traversed arcs costs ($cost(r) = \sum_{p=0}^{s} c_{i_p, i_{p+1}}$) and it is feasible if is respected the maximum vehicle capacity ($q(r) = \sum_{i=1}^{s} q_i \leq Q$).

A solution to the *CVRP* is a set of routes $S = (r_1, r_2, ..., r_k)$ and is feasible if all its routes are feasible and they induce a partitioning on N. A feasible solution will be evaluated with the number of used vehicles k and with the total travel cost $cost(S) = \sum_{i=1}^{k} cost(r_i)$. Figure 2.1 graphically shows the representation of a *CVRP* problem and one of its possible solutions.

2.1.2 Vehicle Routing Problem with Time Windows

The *Vehicle Routing Problem with Time Windows* (VRPTW) is a generalization of the previously presented *CVRP*, where the service at each customer has to start within a defined time interval, called time window. The *VRPTW* has emerged as an important area for progress in handling realistic complication and generalization of the basic routing model[42] and introduces a set of other variants, all based on time constraints.

The time window is represented by the pair e_i, l_i, that respectively define the minimum and the maximum time instants for the service start. It's important to note that the time constraints don't refer to all the service time and don't imply the service termination within the l value.

The *VRPTW* model defines the graph's vertex set as $V = \{0, n+1\} \cup N$ where 0 and $n+1$ represent the depot as source and sink vertices. The time window associated with the depot $[e_0, l_0] = [e_{n+1}, l_{n+1}]$ represents the earliest possible departure time and the latest possible arrival time at the depot for all the vehicles.

Time windows could be hard or soft: in the first case if a vehicle arrives too early to a certain costumer, it has to wait until the ready time defined for that point (usually without waiting costs) and if arrives to late it will not serve the customer, in the second case every violation of the time windows will introduce

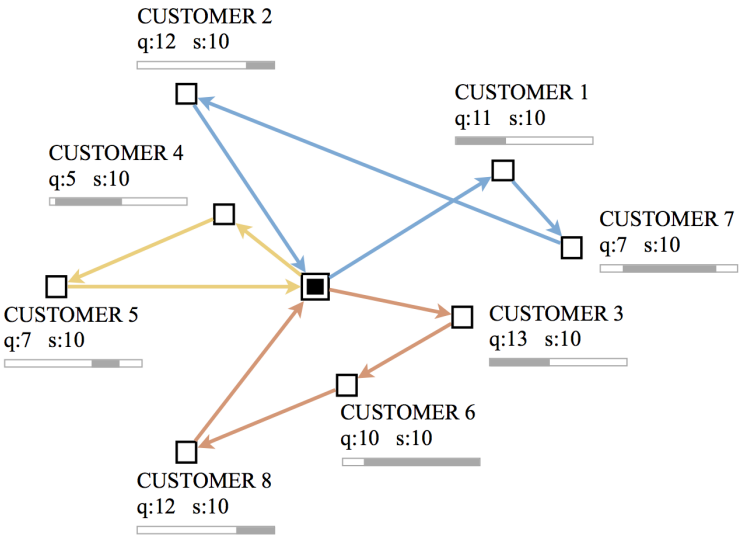


Figure 2.2: *VRPTW* representation. *Source: author*

a penalty cost, according to specific penalty functions defined for the context, without stopping the service.

Moreover time windows could be divided into one-sized, that means that it's defined only the ready time and not the due time, and two-sized, using both the time constraints. In the present document, two-sided hard time windows will be considered.

From a computational point of view, time windows could be distinguished between tight and loose. The literature refers to tight time windows when them influence the solution representing an active constraint, in the other case the loose time windows don't cause variation, falling back to a simple *CVRP* solution. Another distinction could be done between narrow and wide time windows: in this case the definition is based on the relative window width compared to the planning horizon. It's important to remark that a narrow time window is not necessarily tight at the same time.

The last consideration would be on the possibility to use spatial constraints in order to solve the *VRPTW*. In the *CRVP*, the geography is usually the important factor determining the routes plan, and a solution based only on the customer positions would likely be an acceptable starting point for a local search or genetic evolution. On the contrary, in the time window variant, routes are usually con-

strained by the vehicle capacity or the availability time interval of the customers. This fact increases the difficulty to find a good delivery plan making the problem manually infeasible even with just a few customers. In that context, spatial driven solutions could bring to plans really far from the optimal one, making them unattractive even in order to quickly reach an initial approximation.

2.1.3 Mathematical formulation

In the following section, starting from the previous definition of the problem, will be presented an integer programming formulation for *VRPTW*.

In the model are introduced two types of variables: x_{ijk}, a binary variable that is equal to 1 if the arc (i,j) is used by the vehicle k, 0 otherwise, and T_{ik} that is the service start time at vertex i by vehicle k.

$$minimize \sum_{k \in K} \sum_{(i,j) \in E} c_{ij} x_{ijk} \tag{2.1}$$

$$\sum_{k \in K} \sum_{j \in \delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N \tag{2.2}$$

$$\sum_{j \in \delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K \tag{2.3}$$

$$\sum_{i \in \delta^-(j)} x_{ijk} - \sum_{i \in \delta^+(j)} x_{ijk} = 0 \qquad \forall k \in K, j \in N \tag{2.4}$$

$$\sum_{i \in \delta^-(n+1)} x_{i(n+1)k} = 1 \qquad \forall k \in K \tag{2.5}$$

$$x_{ijk}(T_{ik} + s_i + c_{ij} - T_{jk}) \leq 0 \qquad \forall k \in K, (i,j) \in E \tag{2.6}$$

$$e_i \leq T_{ik} \leq l_i \qquad \forall k \in K, i \in N \tag{2.7}$$

$$\sum_{i \in N} q_i \sum_{j \in \delta^+(i)} x_{ijk} \leq Q \qquad \forall k \in K \tag{2.8}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in E \tag{2.9}$$

The objective function (2.1) minimizes the total cost of the solution, the set of constraints (2.2) ensures that each customer will be served by exactly one route, constraints (2.3)-(2.5) define the path from the source to the sink for all the vehicles. Constraints (2.6) guarantee that the service starting time at customer j is feasible considering the service time of the previous one and the time needed to go from a costumer to the other. Constraints (2.7) verify that each arrival time has to fall in the respective time window. Constraints (2.8)

guarantee the capacity feasibility of the route and finally, (2.9) forces the binary value of the variables x_{ijk}.

How can be seen from the equation (2.6), the model is nonlinear. The literature[20] suggests some methods to linearize the previous model, but for our work, this formalization will be useful only for a clearer comprehension of the problem.

2.1.4 Benchmarking

Due to the large number of proposed solutions for the $VRPTW$, a set of standard tests[15] is required in order to evaluate and compare the results of suggested heuristics and meta-heuristics, being, usually, impossible the computation of the relative exact optimal result.

In 1987, Solomon introduced a set of 56 benchmark problems, that is still the leading test for $VRPTW$ evaluation. The researchers need for bigger problems, closer to real world problems, made Gehring and Homberger propose, in 1999, a series of extended Solomon problems, allowing benchmark until 1000 customers.

In order to compare results with the ones proposed by the literature, it's important to remember that for the exacts methods the objective function of the problem is the minimization of the total routing cost, but for heuristics methods, it usually considered firstly the minimization of the vehicle number and secondly the minimization of the traveling costs.

Solomon instances

As already mentioned, the Solomon instances[43] are composed by 56 problems, each one available with 25, 50 or 100 customers. The sets reflect several structural factors in vehicle routing and scheduling such as geographical data (the customers are distributed within a 100x100 square area) and characteristics of the time windows, like tightness positioning and the fraction of time-constrained customer in the instances.

The instances are divided in 3 classes: C1 and C2 have customers located in clusters, in R1 and R2 the geographical datas are randomly generated by random uniform distribution, RC1 and RC2 contain a mix of both random and clustered customers.

The sets of C1, R1 and RC1 problems are characterized by short scheduling horizon, where the length of the route-time constraints acts as a capacity constraint which, together with the vehicle capacity constraints, permits that only

a few number of customers can be served by a single vehicle. In contrast, the sets C2, R2 and RC2 have a long scheduling horizon and a large vehicle capacity that makes the problems very hard to solve exactly and this fact turns the time windows the real constraint for the solution.

Homberger and Gethring instances

Due to the increasing quality of the state-of-art solutions and to the need to compare the proposed methods with large-sized instances, more similar to the real world problems, where the huge number of clients turns the usage of exact methods impracticable, in 1999 Homberger and Gethring[18] proposed a set of 300 problems divided in 200, 400, 600, 800 and 1000 customers.

Each group is divided in 6 classes, the same ones proposed by Solomon, then representing an extension of the original sets. Each category preserve the original characteristic in terms of customer allocation and time windows assignment.

2.2 Computational complexity

Dealing with complex problems, one of the primary key aspects is the possibility to achieve a result within an acceptable time interval, providing an efficient solution, viable in real contexts. The theory of computational complexity is used in mathematics and in computer science to determine the effort that is needed in order to complete the execution of an algorithm.

The algorithm complexity is the estimation of the ratio between the size of the input problem (a characteristic dimension of the datas, like the number of entities that are involved in the problem) and the quantity of used resources. Usually an algorithm is evaluated based on time complexity (execution time required to solve the problem) and space complexity (size of used memory in order to reach the solution).

For extension, the complexity of a problem (or a class of problems) is defined as the complexity of the most efficient algorithm able to solve the problem.

Based on the previous definitions, all the problems are divided into complexity classes, starting from the constant one, where the running time doesn't increase with the growth of the problem size, passing by linear, polynomial, exponential, until factorial complexity. Moreover, in the computational complexity

theory, has to be considered the nondeterminism, that means that the algorithm can exhibit different behaviors on different runs.

A notable category in these terms is the NP-complete class. The abbreviation NP refers to "nondeterministic polynomial time" and means that only a nondeterministic machine could find the optimal solution in polynomial time. It's important that given a solution to an NP-complete problem, it could be verified in polynomial time.

Nowadays, NP-complete problems represent a challenge for computer science community. Considering real world applications, with very large-sized instances, the usage of exact methods to compute a solution of these kind of problems is unfeasible. For this reason heuristic methods have to be applied in order to achieve a good sub-optimal solution.

In this sense, imaging the enumeration of all the feasible solutions (theoretically possible), it's necessary to develop methods capable of evaluate some problem characteristics to examine the solutions' domain avoiding the exhaustive browsing of all the possibilities.

Since the *CVRP* is NP-hard, being a generalization, the *VRPTW* is also NP-hard, as proven by Lenstra[28]. In fact even finding a feasible solution to the *VRPTW* for a fixed number of vehicle is itself an NP-complete problem[41].

2.3 Randomic algorithms

Usually the computation of a complex algorithm is performed in order to minimize some operational costs, trying to find the best possible solution to maximize profits. However, in many cases, the saving due to the detection of the optimal solution, compared with a good approximation, doesn't justify the huge running time required by NP-complex algorithms to compute this minimum, since the number of possible solutions usually grows with a order of $n!$ or n^n. Furthermore, the speed required by modern services and applications simply turns infeasible the usage of an exact algorithms to solve these NP-problems.

In these cases, a faster algorithm able to provide a sub-optimal solution may be preferred in order to approximate the unreachable exact solution.

In the NP-complete problem therefore, the choice between solution quality and execution time is a threshold point that depends on the application field and the service requirements.

A randomized (or probabilistic) algorithm is based on the idea of randomize some choices during the computation in order to avoid the analysis of all the possible branches. This behavior lets to reduce the execution time, performing well in the average case and minimizing the occurrences of best and worst cases[17]. In fact, generally a probabilistic algorithm has a better average-case complexity.

The randomized algorithms could be divided in three categories according to the type of produced output.

- Numerical algorithms provide an approximated result with some confidence, improving precision at each algorithm iteration.
- *Las Vegas* algorithms, that exploit randomness to reduce memory and time usage in the computation procedure, but still guaranteeing the achievement of the optimal solution.
- *Monte Carlo* algorithms, that ensures only probabilistic guarantees on the generated solution quality.

2.4 Genetic algorithms

In computer science, a genetic algorithm is a search meta-heuristic that mimics the natural selection process. The terms "meta-heuristic" is due to the fact that it doesn't represent a solution of a specific problem, but defines a framework and a methodology to solve a large variety of optimization problems.

Genetic algorithms start from the generation of an initial population of possible acceptable solutions and apply techniques inspired by natural evolution, such inheritance, mutation, selection and crossover in order to produce a suboptimal solution starting from the initial ones. The basic structure of a genetic algorithm is the following: starting from a pool of solutions representing the initial generation, two individuals are iteratively selected according to their quality (the best individuals are selected with an higher probability) and combined using a crossover operator that mix them using specific procedures to produce a offspring that combines the parents characteristics. Then a mutation could happen, randomly changing some alleles. Applying this reproduction strategy several times are obtained new individuals that will compose the next generation. At each iteration of the algorithm every solution of the population pool

Algorithm 2.1 Genetic algorithm

1: $P \leftarrow$ initialize-population()
2: **while** stopping conditions are not satisfied **do**
3: $p_1, p_2 \leftarrow$ selection(P)
4: $p_{new} \leftarrow$ crossover(p_1,p_2)
5: $p'_{new} \leftarrow$ mutation(p_{new})
6: $P \leftarrow$ update-population(P,p'_{new})
7: **end while**
8: **return** best-solution(P)

could be used as final result. Algorithm 2.1 shows the basic genetic algorithm structure.

A function called *fitness*, is previously defined to compute a value that should represent the quality of a given solution, the goal of the evolutionary procedure is combine some partial "good qualities" of the previous generation individuals to generate a better offspring.

This meta-heuristic has prove[19] to be a powerful and broadly applicable stochastic search and optimization technique that really works for many problems that are difficult to solve by conventional methods.

As will be presented in the next chapter, genetic algorithms has found an application field in vehicle routing problem, too, motivating many interesting researches in that area and resulting a really good methodology in order to generate a fleet routing plan. In the present work will be not treated in deep this kind of algorithms, but in the next section will be presented some considerations that have driven the current work in order to use the proposed solution to generate the initial population for a $VRPTW$ genetic algorithm.

2.4.1 Diversity control

The applications of genetic algorithms in combinatorial problems are frequently affected by an early convergence problem and the evolutionary processes are often trapped in local optima. This premature convergence occurs when the population of a genetic algorithm reaches a suboptimal state and the genetic operators can no longer produce offspring with a better performance than their parents. For this reason, a diversity control technique is required in order to prevent this problem and then improving the final solution quality[32].

The previous consideration unfortunately comes into conflict with another

fundamental observation: the selection of the initial population in a population-based heuristic optimization method is important, since it affects the search for several iterations and often has an influence on the final solution[31]. Usually, however, it is more important that the solutions are distributes as much as possible and they are randomly initiated.

Therefore, programming a new genetic algorithm, should be evaluated a trade-off between the need of a completely random generation that guarantee the most sparse population and the usage of domain knowledge in order to produce a better pool population which, however, may be afflicted by local optimum premature convergence.

Despite the importance of this initial step is widely recognized[26, 27], in the $VRPTW$ genetic algorithm context, that point is not enough analyzed and many times a simple random generator is used[45, 46, 30].

Chapter 3

Literature Review

This chapter presents a survey of the research on the vehicle routing problem with time windows. In order to proceed presenting a new heuristic, we will provide a brief review on existing solutions and current state-of-art. In addition to an exhaustive analysis of the constructive heuristics, are explained some important concepts, as the infeasible solutions approach and the giant tour representation, used by the modern meta-heuristics like local searches or genetic algorithms. This part will be fundamental to better understand the state of the current research and of the best reached results.

The following sections firstly report a really short review of the exact methods, that are not the focus of our work, then the most relevant constructive heuristics are presented, explaining the basic concepts of the proposed methods. The last section describes some important modern meta-heuristics that provide the current best known solutions.

3.1 Exact methods

Since the formal definition of the problem in 1987, numerous exact methods have been proposed for the *VRPTW*. A review of all the algorithms developed until 2000 will be provided by Cordeau et al.[9]. The recently developed solutions could be classified in three categories: Branch-and-Cut-and-Price[23], Branch-and-Cut[25] and reduced set partitioning[3].

In order to understand the real complexity of *VRP* and how difficult the application of exact algorithm could be, you should consider that until Jopsen et al.[23] paper publication, only 35 of the 56 instances of the Solomon 100-customers set were solved to optimality. In 2011, Baldacci, Migozzi and Roberti[3] were able to solve all instances except one due to a lack of memory. One year later, Ropke[40] reported to be able to solve all the Solomon instances with a Branch-and-Cut-and-Price algorithm but until now no technical paper describing the algorithm is available.

Looking at experimental datas obtained applying exact methods, as reported in [45], it could be noted that running the Solomon's instances, those methods shows an higher difficulty solving the class 2, because wide time windows increase the number of feasible routes and the number of customers per feasible route.

It's expected that with more computational power and optimized methods, in the future should be possible approach the Gehring and Homberger benchmarks, but until now really few experiments was performed in that direction, and considering the huge effort done to solve the small 100 problem set, it should be stated that exact methods never will represent a competitive solution for big-sized real applications.

In this work will not be presented in details the algorithms of the exact solutions because it's not the focus of this thesis and don't have a direct impact on the developed heuristics solutions. For more informations about these methods we have reported all the sources of the most relevant works.

3.2 Heuristics

Heuristics are solution methods that can often find good-quality feasible solutions relatively quickly. However, there are no guarantees regarding solution

quality. Because the already presented limitations of the exact methods, heuristic algorithms have driven the modern research in *VRP* area providing solutions for big-sized real problems with hundreds or thousands of customers.

Formally all the methods that are based on decisions taken using estimation formulas are considered heuristic methods, but usually they are divided into construction heuristics and meta-heuristics. In this section we will presents three of the most used and relevant constructive heuristics.

According to [7], constructive heuristics are algorithms that select nodes (or arcs) sequentially until a feasible solution has been created. Nodes are chosen based on some cost minimization criteria, often subject to the restriction that selection doesn't cause violations of vehicle capacity or time window constraints.

The constructive heuristics are commonly divided in two categories: sequential methods, that define one route at time adding all the possible customers to the current vehicle, and parallel methods, that proceed defining concurrently the scheduling for all the involved vehicles.

Solomon M. (1987)

In 1987 Solomon proposed his original work[43] about algorithms time constrained vehicle routing problems. This paper will give the start of a deep and wide research on *VRPTW* problems. As reported by the same Solomon, at the end of 80s, the spatial problem of routing vehicle had been intensively studied in the literature, but only few secondary works were done on *VRPTW*. The limitation was almost computational, resulting really hard solve with the current strategies instances where customer service was time-constrained.

His belief that these approaches suffered a huge limitation not considering time windows, brings him to propose a new set of benchmark problems and a new algorithm for the resolution of this variant of the simple *VRP*.

In [43], after a brief introduction about current techniques for *VRPTW*, Solomon presented the so called *Push Forward Insertion Heuristic* (PFIH). As the name itself explains, the algorithm is based on a insertion strategy, oriented to minimize the push forward costs in the current partial route, where for push forward is intended the operation to postpone an already scheduled service to turn possible the attendance of the inserted customer.

The *PFIH* is a sequential heuristic that initializes a new route with a cus-

tomer using a given criteria, then iteratively chooses the unserved customer with the minimum push forward insertion cost and repeats these insertions until possible. When a route is builded a new vehicle is initialized choosing another first customer within the unserved set.

To choose the customer to be inserted, two cost functions (c_1 and c_2) were defined to compute the insertion cost in all the possible positions for each available customers. Let $(i_0, i_1, i_2, ..., i_m)$ be the current partial route, with $i_0 = i_m = 0$. For each unrouted customer u, it first computed its best feasible insertion place.

$$c_1(i(u), u, j(u)) = \min \left[c_1(i_{p-1}, u, i_{p)} \right], \ p = 1, ..., m \tag{3.1}$$

Then the customer to be inserted is computed selecting the best one within the previous computed (at chosen position).

$$c_2(i(u^*), u^*, j(u^*) = \text{optimum} \left[c_2(i(u), u, j(u)) \right], \ u \text{ feasible} \tag{3.2}$$

Customer u^* is the chosen customer to be inserted in the route between $i(u^*)$ and $j(u^*)$. In the original paper, Solomon reports three different insertion cost formulas (c_1, c_2 pairs), called *I1*, *I2* and *I3*. As demonstrated by the author the first one provides the best results on the majority of the instances and in all the following literature, this one was adopted as standard insertion cost estimator for *PFIH*. In the next paragraphs will be briefly presented the *I1* version.

$$c_1(i, u, j) = \alpha_1 c_{11}(i, u, j) + \alpha_2 c_{12}(i, u, j) \qquad \alpha_1 + \alpha_2 = 1 \tag{3.3}$$

$$c_{11}(i, u, j) = d_{iu} + d_{uj} - \mu d_{ij} \qquad \mu \geq 0 \tag{3.4}$$

$$c_{12}(i, u, j) = b'_j - b_j \tag{3.5}$$

$$c_2(i, u, j) = \lambda d_{0u} - c_1(i, u, j) \qquad \lambda \geq 0 \tag{3.6}$$

The $c_1$cost in equation 3.3 is computed as combination of two different costs. The two parameters α_1 and α_2 are used as weight to choose the two different estimator c_{11} and c_{12}. The first one (equation 3.4) follows a saving strategy measuring of the required additional travel cost to serve the customer. The second one is the push forward cost for the following customer, that measures the difference between the estimated service start before and after the insertion.

In his work, Solomon tested four configurations $(\mu, \lambda, \alpha_1, \alpha_2)$: $(1, 1, 1, 0)$, $(1, 2, 1, 0)$, $(1, 1, 0, 1)$ and $(1, 2, 0, 1)$. Experimental results executed on different configurations didn't have highlighted relevant differences and the author has

not provided ways to previously estimate the most suitable configuration.

All the results reported in the original paper refers to the best solution obtained testing all the four configurations combined with two different criteria for the route initialization choice: the farthest unrouted customer and the unrouted customer with the earliest dead-line. According to the published results, the reached CVN was 453.

In 2000, Dulleart[16] has reported that Solomon's time insertion c_{12} cost underestimates the time needed to insert the new customer at the first position of the partial route. The introduction of new criteria offer significant cost savings, but the impacts of the saving decrease as the number of customers per route increases.

Potvin JY. and Rousseau JM. (1993)

In 1993, Potvin and Rousseau published a parallel version[38] of Solomon's insertion heuristic *I1*. The new algorithm made use of the generic Solomon's framework, redefining the route initialization procedure and some insertion cost formulas to generate and build more routes simultaneously.

To initialize the n_r routes, the first challenge was define the best number of necessary vehicles. To do that Potvin run a single execution of the sequential algorithm with a standard configuration. Once obtained the *PFIH* solution, the routes are initialized using as first customer the last one of each previously computed route. Starting from this partial results, each customer is evaluated computing the best insertion position on all the available routes and the one with the lower cost is selected for the next insertion following the Solomon's framework.

$$c_{1r}^{*}(i_r(u), u, j_r(u)) = \min_{p=1,...,m} \left[c_{1r}(i_{r_{p-1}}, u, i_{r_p})\right], \ r = 1, ..., n_r \tag{3.7}$$

$$c_{1r}(i_r, u, j_r) = \alpha_1 c_{11r}(i_r, u, j_r) + \alpha_2 c_{12r}(i_r, u, j_r) \tag{3.8}$$

In the first step, for each customer is computed the best feasible insertion place in each one of the n_r routes (equations 3.7 and 3.8). As could be seen, for the computation of c_1 the used formulas are the same of the Solomon's solution, just adapted to work on several routes. As c_{11} and c_{12} costs are adopted the same methods of *PFIH*.

Then c_2 costs are computed selecting the customer with the better estimation. Equations 3.9 and 3.10 reports the used formulas that differ from the *PFIH* ones . $c^*_{1r'}$ refers to the best route insertion cost for a given customer.

$$c_2(u^*) = \max_u [c_2(u)] \tag{3.9}$$

$$c_2(u) = \sum_{r \neq r'} [c^*_{1r}(i_r(u), u, j_r(u)) - c^*_{1r'}(i_{r'}(u), u, j_{r'}(u))] \tag{3.10}$$

For the customer choice, Potvin introduced a new strategy. As could be seen looking equation 3.10, the metric used to choose which is the next customer that should be inserted is a look-ahead regret estimation. Generalizing the idea proposed by Tillman and Cain[44], the summation of all the differences between the best alternative and the others represents the urgency of the customer insertion. In fact an high value means that the other alternatives are not so good and that the current one should be chosen preventing future forced bad insertions, due to the possible infeasibility of that solution. This methodology comes from the CSP (constraint satisfaction problem) optimization, where variables with less high quality assignments have an higher priority.

Moreover, Potvin proposed an iterative procedure that reduces of one the number of initial routes running the algorithm until feasible solutions are reached, trying to minimize the number of required vehicles.

In the experiments executed in the original paper, three different configurations are used to find the best solution: $(\alpha_1, \alpha_2) = (0.5, 0.5), (0.75, 0.25), (1.0, 0.0)$. According to the published results, the reached CVN was 453, the same value of *PFIH*, but obtaining better computational performance. It's important to note that these better performances refers to the total execution time required to test all the configuration set, both for *PFIH* and Potvin algorithms. Considering only one run of the procedures, the Potvin solution obviously requires higher times because the *PFIH* resolution is alway a partial result of the entire method.

Ioannou G. et al. (2001)

The last constructive heuristic that would be presented in this chapter was proposed by Ioannou et al. in 2001[22]. The algorithm proposed in their paper is an adaptation of the framework proposed by Solomon based on new criteria for

customer selection and insertion that exploits the minimization function of the greedy look-ahead solution approach of Atkinson[4].

This method uses different insertion cost formulas to compute a deeper analysis of the real impact caused by an insertion on customers already served by the evaluated vehicle, on customer that are still unrouted and on the time window of the customer u, himself.

$$IS_u = b_u - e_u \tag{3.11}$$

Equation 3.11 represents the own impact property, that measures the time passed between the beginning of the time window and the effective vehicle arrival time at customer u. Low values of this metric mean an higher compatibility with other possible insertions.

The second property, the external impact IU_u, is oriented to minimize the restrictions for the other unrouted customers insertion. Performing an insertion, the time windows overlapping of scheduled and unrouted customers increases, becoming harder find a feasible insertion.

$$IU_u = \frac{1}{|J| - 1} \sum_{j \in J - \{u\}} \max\left\{(l_j - (e_u + d_{uj})), (l_u - (e_j + d_{ju}))\right\} \tag{3.12}$$

Equation 3.12 shows the used formula for external impact, where J is the set of all unrouted customers.. It's important to remember that customer u could be inserted before or after j, then only one of the two values of the max function is positive.

The last property, the internal impact IR_u, measures the impact on the customers already scheduled in the current route. This values is a combination of three aspectS: the travel cost increase, the push forward cost and compatibility of the selected customer time window with the specific insertion place.

$$c_{1u}(i,j) = d_{iu} + d_{uj} - d_{ij} \tag{3.13}$$

$$c_{2u}(i,j) = [l_j - (a_i + s_i + d_{ij})] - [l_j - (a_u + s_u + d_{uj})] \tag{3.14}$$

$$c_{3u}(i,j) = l_u - (a_i + s_i + d_{ij}) \tag{3.15}$$

Equations 3.13 - 3.15 report the previously presented specific costs. The linear combination (equation 3.16) of these three factors composes the local disturbance property, that is the impact due to the insertion of u in a certain

point of the route. The internal impact, also called global disturbance (equation 3.17), is then obtained as the average of the local disturbance factor on all the possible insertion position.

$$LD_u(i,j) = b_1 c_{1u} + b_2 c_{2u} + b_3 c_{3u} \tag{3.16}$$

$$IR_u = \frac{1}{|I_r|} \sum_{(i,j) \in I_r} LD_u(i,j) \tag{3.17}$$

Defined the three components of the insertion impact, the insertion cost is calculated combining them (equation 3.18). The obtained value is used to choose the next customer to insert in the route under construction. As can be seen this value takes into account all the aspects already presented by Solomon with his *PFIH*, introducing other considerations about the minimization of the constraints that a choice implies on the future steps.

$$Impact(u) = b_s IS_u + b_e IU_u + b_r IR_u \tag{3.18}$$

Using the proposed heuristic, and running several configurations of the parameters, Ioannou obtained a total CVN on the 56 instances of 429 vehicles. According to the Bräysy and Gendreau's literature survey[7], the total time required by the *Impact* algorithm to solve all the instances is approximately 4 minutes (time for a single run), while the reported time required by *PFIH* was 0.6 minutes.

3.3 Metaheuristics

As already introduced in the previous section, in computer science heuristics methods are divided into "pure" heuristics and metaheuristics. While heuristic methods, like the ones reported above, are algorithms designed expressly to solve a certain problem, applying methodologies specifically adapted for the application field, for metaheuristics is intended a general mathematical optimization algorithm designed to scan the problem domain following defined criteria.

Usually the first step to use a meta heuristic is define a problem codification, according to the heuristic behavior, then some specific operators such fitness functions or neighbors generators should be specified and the metaheuristic applies well studied strategies to browse the possible solutions searching for an optimal result.

The first example of a metaheuristic is the local search, a simple strategy that works looking only at one portion of the search domain, optimizing the result in that subset and then moving towards that solution applying iteratively the same procedure in another portion. Other two well knowns metaheuristics, that were nature-inspired, are the genetic algorithms and the ant-colony optimization. The first one tries to reproduce the biological evolution mechanism, generating offsprings based on the best characteristics of the parents, preserving the good partial solutions. The second is based on the swarm-intelligence method derived from the analysis of the ants behavior, that tends to reinforce a route when they find a good food source.

In this section we will summarize the most relevant works on $VRPTW$ that make use of metaheuristic methods, just to provide a better overview of the current state-of-art of this area. Firstly a brief explanation of the most used local search techniques is presented to show how could be defined operators for vehicle routing problems, then are reported some memetic algorithm applications that today provide the best known solutions.

3.3.1 Local search

Local search, as already anticipated in the introduction, is a metaheuristic based on the neighborhood concept. Let $\mathscr{P}(S)$ be the set of solutions subsets in S. We define a neighborhood function as a function $N : S \to \mathscr{P}(S)$ that maps from a solution s to a subset of solutions $N(s)$. This subset is called the neighborhood of s. A solution is said to be locally optimal if the cost function computed in s is less than for all the other values in $N(s)$.

One defined the concept of neighborhood, a local search algorithm works, in general, starting from a random solution and computing its neighborhood set, then the best solution in this set is chosen to iterate another time the procedure. The algorithm stops when it's not able to find better solutions, this means that it has reached a local minimum.

Starting from the basic definition, there are different ways to guide the local search algorithms. In fact the application of the simple procedure is usually affected by premature convergence to local minimum and in general these metaheuristics works fine to analyze in depth a certain subdomain but it's not naturally suitable to scan in breadth the whole domain. Tabu search and simulated annealing are two of the most used techniques.

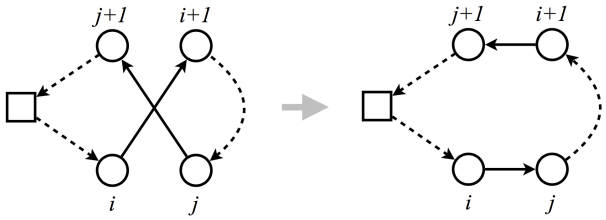


Figure 3.1: Illustration of the 2-opt neighborhood. *Adapted from: Toth*[45]

The first one introduces a relaxation of the basic structure allowing movings towards worst solutions if the algorithm is no more able to improve the solution. These jumps are accepted to try to exit from local minimum, moreover tabu search introduces prohibitions to discourage the search from coming back to previously-visited solutions.

Also the second technique is based on the consent of worsening movements. Taking inspiration from the metallurgic treatment, it's defined a variable, called temperature, initialized with an high value, that decreases at each iteration of the local search algorithm. The probability of accepting a worse solution is then proportional to the temperature value. This means that, at the beginning, the algorithm tends to prefer a search in breadth, analyzing a large domain portion, and, proceeding with the search, the heuristic starts to focus on a specific interval performing an in-depth optimization.

In the following paragraphs we will present the most used neighborhood functions in *VRPTW* field, showing how two routing plans could be defined close. The last part is dedicated to a brief presentation of the most relevant works that used a local search strategy to find the optimal solution. Moreover these concepts will be fundamental to better understand the hybrid algorithms that will be presented in the next section.

2-opt Neighborhood This is an intra-route neighborhood, that means the solutions contained in the set differ from the original one just for the order of customers served in one route. Using 2-opt two arcs in a route are selected, replacing them with two other arcs to reconnect the route and changing the orientation of the subpath that doesn't contain the depot.

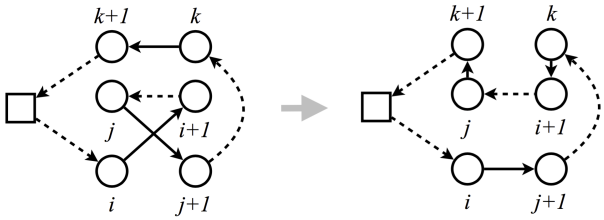


Figure 3.2: Illustration of the Or-opt neighborhood. *Adapted from: Toth*[45]

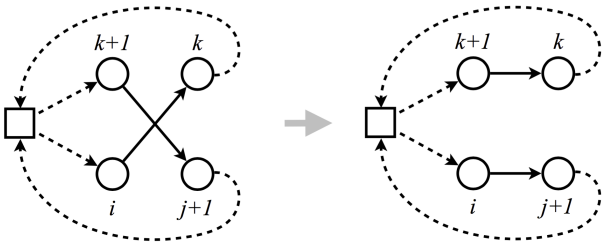


Figure 3.3: Illustration of the 2-opt* neighborhood. *Adapted from: Toth*[45]

Or-opt Neighborhood This one is an intra-route method, too. Or-opt generates the neighborhood relocating a subpath in another place of the route, preserving the relative order. Usually are considered only paths with a limited length. Bräysy[6] proposed a variant called IOPT where the subpath is reversed before the reinsertion.

2-opt* Neighborhood This neighborhood is the inter-route version of the one already presented. While intra-route means a modification of a single route, inter-route operations imply an arcs interchange across two different routes of the same solution.

In this case two arcs in different routes are deleted reconnecting them with two new arcs without change the relative order of the partial subpaths.

Cross Exchange Neighborhood This inter-route neighborhood is defined selecting two subpaths on different routes, usually with a limited size, and then replacing them exchanging their position across the two routes. This method could be used also in an intra-route fashion where the subpaths to be exchanged belong to the same route.

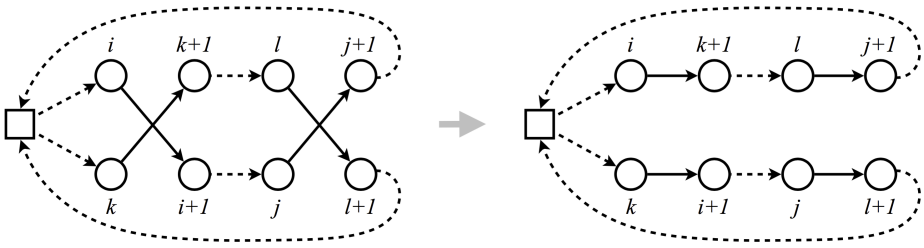


Figure 3.4: Illustration of cross exchange neighborhood. *Adapted from: Toth*[45]

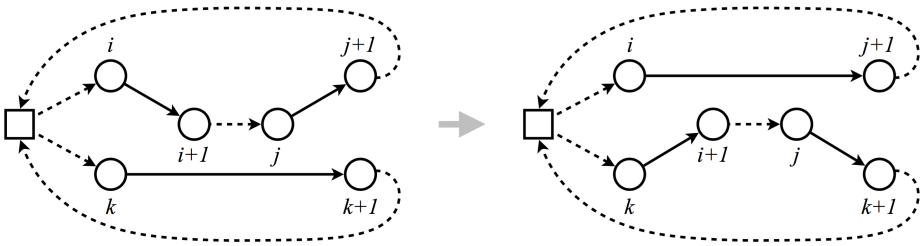


Figure 3.5: Illustration of path relocation neighborhood. *Adapted from: Toth*[45]

Path Relocation Neighborhood Contrary to the previous methods, in this inter-route technique is selected only one subpath from a route and relocated pushing it in another one. This neighborhood could be seen as a special case of the Cross Exchange methods, where one of the two paths is always empty.

λ-Interchange Neighborhood The lambda-interchange is a generalization of the cross exchange neighborhood proposed by Osman and Christofieds[37]. Given a feasible solution for the VRPTW represented by $S = \{..., R_p, ..., R_q, ...\}$, where $R_p$is a set of customers serviced by a vehicle route p. A λ-interchange between a pair of routes R_p and R_q is a replacement of subset $S_1 \subseteq R_p$ of size $|S_1| \leq \lambda$ by another subset $S_2 \subseteq R_q$ of size $|S_2| \leq \lambda$. The neighborhood $N_\lambda(S)$ of a given solution S is the set of all neighbors $\{S'\}$ generated by the λ-interchange method for a given λ.

In local search moreover, two different strategies could be applied to perform a movement: the *First-Best* (FB) strategy, that will select for the next iteration step the first neighbor founded with a lower cost estimation, and the *Global-Best* (GB) strategy that analyze all the neighborhood before taking a decision.

Local search solutions have represented the state-of-art for many years providing the best know solution, until the introduction of memetic algorithms, an

hybrid technology where local searches still play a fundamental role. Just to give a brief overview of these methods applied to vehicle routing problems, we will list some published works.

As already mentioned, in 1989 Osman and Christofieds[37] have proposed an heuristic based on simulated annealing and defining the solutions neighborhoods with the λ-interchange method.

In 2004 Bent and Ban Henteryck[5] published a two-stage hybrid local search heuristic combining a simulated annealing phase for the vehicle minimization with a large neighborhood search (LNS) to minimize the secondary cost objective function. The LNS is a technique based on the idea to generate a new solution removing a pool of already routed customers replacing them using a constraints oriented decision three. In their work, the neighborhood operator is randomly chosen each time between the ones proposed above.

Ibaraki et al.[21] have proposed in 2008 a classical Iterated Local Search using a first best strategy applied on the following order of neighborhood operators: iopt, 2-opt, 2-opt*, path relocation, cross exchange.

Finally, in 2012, Cordeau and Maischberger[11] used a tabu search strategy, based on a path relocation method, to approach the *VRP* with time windows.

3.3.2 Memetic algorithms

In section 2.4 we have already introduced genetic algorithms, that are one of the evolution based metaheuristics. An evolutionary algorithm firstly generate a pool of feasible solutions in a pseudo-randomic way (or using specific constructive heuristics) and then applies iteratively a set of operators to produce offsprings starting from parents of the previous generation. For that reason EA are also called population-based search methods.

As already mentioned, genetic algorithms are an evolutionary heuristic inspired by the natural genetic that tries to evolve solution for complex problems in the same way the DNA is reproduced in biology.

Memetic algorithms, based on the concept of *meme* introduced by Richard Dawkins in 1976[13], represent a modern variant of the original idea of genetic evolution. As explained by Moscato in [33], this new metaheuristic is an hybrid technology that exploits the ability of GAs to explore in breadth a really huge solutions domain (exploration) and the capacity of the local search to converge in a local optimum given a defined portion of the original domain (exploitation).

MAs have proven to work well for many NP-hard problems such as the *VRPTW*. In the following paragraphs we will report some ideas that have permitted MAs to become the leading solution in this area, defining the new state of art for vehicle routing problems.

Allowing infeasible solutions

Thinking at complex problems like *VRPTW* where time windows and capacity constraints strongly affect the result, it's possible assume that high quality solutions are really close to constraints violation, representing the boundary of the feasible solutions domain. When you design a new metaheuristic, it necessary decide if infeasible solution should be allowed during the algorithm execution.

It's proven that allowing infeasible solution makes it easier to move in the solution space and provide shortcuts between areas of high-quality solutions. However the usage of these techniques turns more difficult the evaluation of the objective function, that should holds in consideration violation penalties and has to ensure that feasible solutions are visited at least occasionally. Despite these drawbacks, allowing infeasible solutions appears to be an important tool to reach high-quality solutions.

Usually three types of infeasibilities are used in *VRPTW* metaheuristics: time windows, vehicle capacity and customer service violations. Typically the first two types are allowed together and the penalized objective function become similar to equation 3.19.

$$\bar{c}(s) = c(s) + \alpha q(s) + \beta w(s) \tag{3.19}$$

Terms $q(s)$ and $w(s)$ are a quantification of the total capacity and time violations for a given solution. In that area a great work was done to define the best strategies to manage the parameters configuration and to compute the penalized values in a constant time. For more informations we suggest the reading of Cordeau, Laporte and Mercier[10] and Nagata[34].

EAX Crossover

Genetic algorithms for *VRPTW* are usually based on solution representations that are permutations of the customers to be served, with or without delimiters. This fact has led to the usage of basic crossover operators defined to combine two strings without assigning any particular interpretation at these datas.

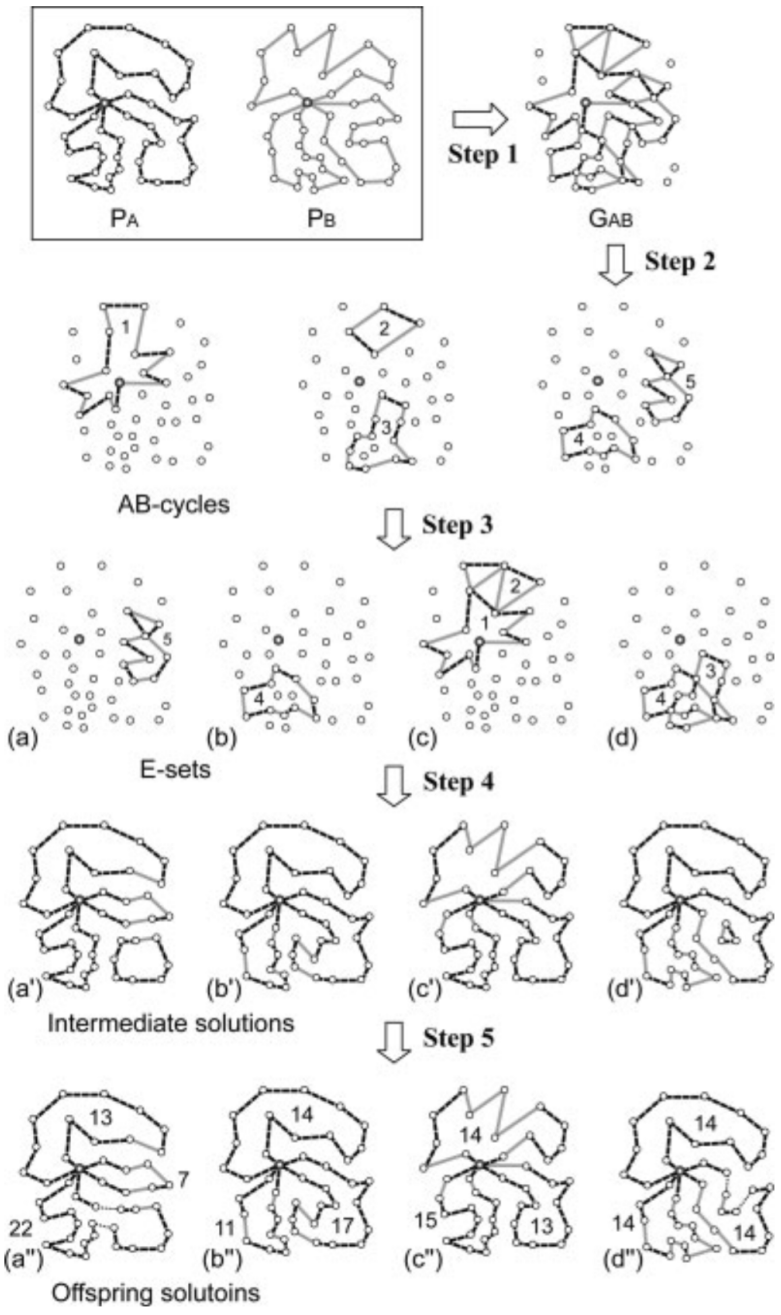


Figure 3.6: Illustration of the EAX crossover steps. *Source: Nagata*[35]

In 1997, Nagata and Kobayashi have introduced a new operator to solve the *TSP* with a genetic algorithm: the EAX crossover. This operator was designed to specifically works on logistic problems where solutions are composed by routing plans. The aim of EAX is combine two different vehicle plans in order to preserve partial solution actuating operations on a routes graph representation and not on meaningless strings.

This new approach has be proven to be the best algorithm for the *TSP*, for that reason it was later adapted to the *CVRP*[35] and, finally, to the *VRPTW* by Nagata, Bräysy and Dulleart in 2010[36].

The EAX crossover algorithm is composed by 5 steps and is based on a graphic representation where each solution is composed by a set of directed arcs that defines the vehicle routes. The first part takes two parents p_A and p_B, and defines a graph G_{AB} composed by all the not-in-common edges.

$$G_{AB} = (V, E_A \cup E_B \setminus E_A \cap E_B) \tag{3.20}$$

In the second step are defined the so-called AB-cycles. An AB-cycle is a typical cycle on G_{AB} graph where edges from p_A and p_B are linked alternately in opposite orientation. When a cycle is found, edges included are removed from G_{AB} graph and the procedure is continued until all edges in G_{AB} are eliminated.

In the third step the E-sets are constructed combining AB-cycles. At the beginning the so-called single strategy is applied, this means that an E-set is composed by a single cycle and this induces only local improvements. After several times without improvements, the algorithm switches to the *block strategy*, where more than one cycle are selected, combining AB-cycles that share at least one customer node. In that latter case we probably obtain a global improvement.

Then, in step four, intermediate solutions are obtained combining one E-set with a base solution (one of the two parents). Considering p_A, the intermediate solution is constructed removing from E_A all the edges in E-set$\cap E_A$ and adding E-set$\cap E_B$. The resulting graph represents the offspring solution, but the result is still not acceptable, because some obtained routes could not include the depot.

In step five the solution is repaired, randomly applying 2-opt* moves to merge unconnected subtours. Figure 3.6 reports the illustration of the EAX crossover steps taken from the Nagata's original paper.

In his paper, Nagata proposes, in addition to the EAX operator, a complete memetic algorithm based on delimitated encoding of the solution, and improved

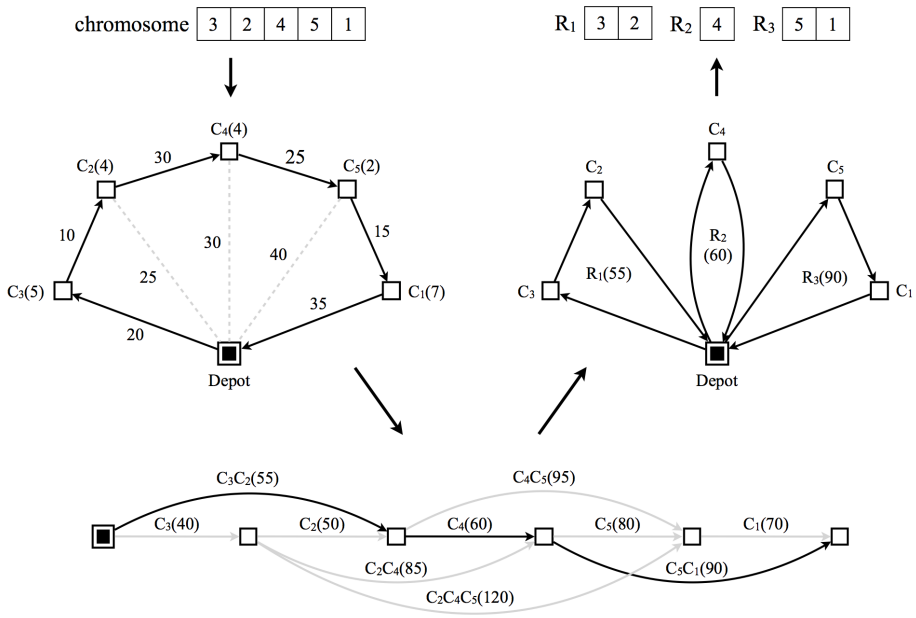


Figure 3.7: Prins proposed split function behavior. *Source: author*

by a local search that uses 2-opt*, cross exchange and path relocation neighborhoods. After its introduction, the EAX crossover was be used by several authors to build new genetic algorithms for *VRPTW*.

Giant Tour Representation

In the solution just analyzed, Nagata had to use a genetic codification for the *VRPTW* solution that explicitly markups different routes. According with the paper published by Xu et al.[50], this kind of representation is a *GR2* encoding, where an individual is a string of customer ids interrupted by vehicle ids that delimit the routes. Another possible method to represent a *VRP* solution inside a genetic algorithm could be the *GR3* one, also known as Giant Tour Representation. In this case an individual is simply composed by a customer ids permutation that has to be divided by a split function in order to generate a *VRP* valid result.

A basic split function could simply read the string adding customers only if constraints are respected and when it's found a customer that doesn't allow the insertion, initiate a new route, until the end of the string.

In 2004, Christian Prins has introduced a more sophisticated split function[39] in order to guarantee the generation of the best routing plan given a certain permutation. This function scans from left to right the chromosome, testing all the possible feasible routes starting from the current customer. If the tested route doesn't violate capacity constraints, it inserts an arc in an auxiliary graph starting from the node preceding the current customer id in the string (0 for the first) to the last customer of the tested route.

At the end of this procedure, the auxiliary graph contains a set of edges that represent all the feasible routes, this allows the computation of the best routing plan simply running a shortest path algorithm. The edge cost can be set to 1 or to the total route cost, respectively if we want to minimize the vehicles number or the total travel cost. Figure 3.7 shows the basic behavior of the Prins split function.

The version proposed by Prins in his paper, works with the basic *CVRP*, but the function can easily extended in order to deal with time constraints how shown by Chang and Chen[8]. The first step is define the service start instant b_{i_j} for customer j in route i.

$$b_{i_j} = \max\left\{e_{i_j}, b_{i_{j-1}} + s_{i_{j-1}} + c_{i_{j-1},i_j}\right\} \tag{3.21}$$

With this definition we obtain that the route starting from the analyzed customer to the *j-th* customer is feasible if all the conditions in equation 3.22 are satisfied.

$$\begin{cases} b_{i_j} \leq l_{i_j} \\ b_{i_j} + s_{i_j} + c_{i_j,0} \leq l_0 \\ \sum_{j \in R_i} q_{i_j} \leq Q \end{cases} \tag{3.22}$$

Equation 3.24 shows how compute the total route cost, considering traveling, waiting and serving costs. This cost function could be also used as secondary objective function for vehicle minimization.

$$W_{i_j} = \max\left\{0, e_{i_j} - \left(b_{i_{j-1}} + s_{i_{j-1}} - c_{i_{j-1},i_j}\right)\right\} \tag{3.23}$$

$$C(R_i) = c_{0,i_1} + \sum_{j=2}^{k}\left(c_{i_{j-1},i_j}\right) + \sum_{j=1}^{k} W_{i_j} + \sum_{j=1}^{k} d_{i_j} + c_{i_k,0} \tag{3.24}$$

HGSADC

In 2013, Vidal et al., have introduced the *Hybrid Genetic Search with Advanced Diversity Control* (HGSADC)[48]. This memetic algorithm has redefined the current state of art for many time constrained *VRP* variants, improving the majority of the best known solutions for both Solomon and Homberger instances. In this section we will briefly present the basic behavior of this methodology.

As defined by the same authors in the published paper, the *HGSADC* algorithm could be defined as

> *[...] a hybrid meta-heuristic combining the exploration capabilities of genetic algorithms with efficient local search-based improvement procedures and diversity management mechanisms. In HGSADC, population diversity is considered as an objective to be optimized along with solution quality through individual evaluations and selections.*

The solution exploits penalized infeasible solutions allowing a time windows relaxation following the lines of Nagata[34], as well the classic load and duration violations. Feasible and not solutions are maintained in two separate subpopulations to maintain an optimal balance between the two.

The chosen encoding for chromosomes is the already presented Giant Tour Representation. In this case the Prins's split function has to be extended allowing edges with penalized cost to correctly deal with the desired infeasibility.

The population diversity is always kept under control using a double ranking strategy: for each individuals are computed two estimators to measure the solution fitness and the diversity contribution, then these values are combined in a general *biased fitness*. When the genetic search starts to be stagnant, the algorithm responds throwing off the worst solutions and executing the reproduction procedure to regenerate the population to the standard size. A deep explanation of the diversity property computation will be presented in section 5.6.

The evolution phase of *HGSADC* is based on a simple Ordered Crossover (OX) for *VRPTW*. This operator generates offsprings preserving a substring of the first parent genotype and filling the missing ones following the relative order of the second parent.

Then the education phase is applied, where local search methods improve the new individual. For this step *HGSADC* applies cross exchange, 2-opt* and

2-opt neighborhood both to reach better results and repair infeasible solutions. In this step an advanced management of the neighborhood exploration is applied to minimize the computation time required to test all the alternatives reducing the total complexity of the algorithm.

The last relevant aspect of this solution is the decomposition phase applied to large instances. This operation is designed specifically for a population-based approach using the informations provided by the best individuals of a generation to geographically and temporally decompose the original problem.

The *HGSADC* was designed to solve not only the basic $VRPTW$ problem but also variants with multiple depots ($MDVRPTW$), multiple periods ($PVRPTW$) and vehicle-site dependencies ($SDVRPTW$). As reported by experimental results, this algorithm provides a competitive solution for all these classes.

Chapter 4

Proposed solution

This chapter will represents the core of this work. The purpose of the proposed algorithm is provide a solution for the *VRPTW* problem presented in the previous chapters. Starting from a problem represented as a Solomon's instance, the aim is identify a sub-optimal solution within a competitive execution time.

The algorithm result is composed by: the used vehicles number, the total service cost and the routing plan with the list and the order of the customers served by each vehicle, this general output allows its adaptation to specific usages and encodings.

The sections of the chapter are organized as follows. The first part will introduce the basic concepts that have driven the algorithm developing, explaining the reasons behind the used structure. Then, in the next section is proposed the full specification of the algorithm, tracing its execution steps on a small sample demonstration. Finally will be presented some details regarding a possible parallel implementation.

4.1 Time Agitation Heuristic

As already mentioned in the previous chapters (section 2.1.2), the construction of a optimal plan in order to serve all the costumers in a *VRPTW* instance is a NP-complete problem. Again, this means that an exhaustive search through all the problem domain, trying to find the best solution, would be unfeasible. Although the literature presents some exacts methods, the application of these procedures is feasible only on some small size problems because increasing the number of customers makes the completion time growing exponentially, preventing the usage of these approaches with real industrial cases.

The complexity of that problem and the size of the existing industrial instances impose the usage of heuristic methods in order to reach a good solution within a reasonable running time.

Starting from a deep research in the field of heuristics methods, this works aims to propose a new constructive algorithm to solve the standard *VRPTW* problem, developing it based on three considerations.

Firstly, time windows constraints represent the primary factor in order to build a new route, the limitations derived from these type of properties are so closed to solve it using a simplified model like the *CVRP*, and introducing only in a second step the time constraints. Therefore the proposed heuristic should be "*time windows driven*" proceeding from the resolution of the time compatibility in order to build a new routing plan.

Secondly, in the field of NP-complete problem solving, the randomized approach has demonstrated to be a effective method to deal with complex problem in a new, simpler fashion, still guaranteeing good results. The idea of trying to apply this concept to the *VRPTW* was inspired by the huge success obtained by previous works applying randomized choices.

The last basic consideration is that we aim to produce a faster algorithm in order to solve even big sized problem in a really quick time accepting a sub-optimal solution, then providing a Pareto-optimal alternative for some application fields.

Our proposed solution represent a completely new approach that define a framework that could be used to solve every other generalization of the *VRP* problem with time windows constraints. However our work is strictly focused on the classical *VRPTW* problem, aiming to minimize primary the number of required vehicles and secondly the total service cost.

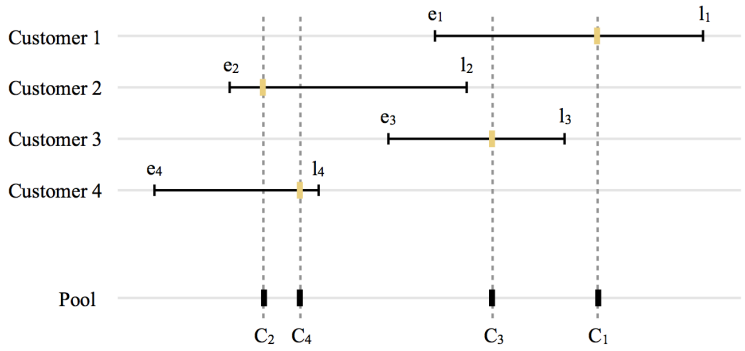


Figure 4.1: *Time Agitation*: customer sorting. *Source: author*

According to the standard literature classification, our Time Agitation Heuristic (TAH) will be labeled as a parallel constructive heuristic, this means that, starting from the instance informations, builds simultaneously several routes to generate an acceptable solution.

In the following section will be given a full explanation of the algorithm presenting the procedure's steps executed in a single run of the algorithm. Obviously, as a randomized algorithm, to obtain a good quality solution, the algorithm has to be run several times according to the level of accuracy required. Details about this and other parameters configurations will be discussed in the next chapter.

4.2 Algorithm specification

Each single run of the *TAH* procedure takes as input the instance object, containing all the informations about the problem, the customers and the available vehicles, and returns a structure containing the constructed plan, specifying for each vehicle which customers has to be served and in which order, the total number of vehicles required and the total service cost, expressed as the total distance covered.

As already mentioned, in order to compare our results with the literature's values, the distance costs are computed as cartesian distances between two points. For real applications it's possible to replace this approximation using a geodetic distance or an external navigation service to estimate the real cost.

Algorithm 4.1 Time Agitation Heuristic

Input: instance
Output: vehicle, cost, routes

1: **for all** $c_i \in instance.customers$ **do**
2: $\quad pool_i \leftarrow id_i$
3: $\quad time_i \leftarrow \text{Random}(e_i, l_i)$
4: **end for**
5: *pool* $\leftarrow$ sort *pool*'s ids accorting to *time* values

6: *routes* $\leftarrow$ ConstructRoutes(*pool*)
7: *vehicles* $\leftarrow$ count routes number
8: *cost* $\leftarrow$ compute total service cost

Algorithm 4.1 shows the pseudocode of the Time Agitation Heuristic. The first step of the procedure (lines 1-4) is select a random instant for each customer, uniformly chosen within its time window's constraints. Figure 4.1 graphically shows the sorting procedure based on the random chosen arrival time.

The idea in this phase is to choose a valid instant that should represent the expected arrival time, like if an imaginary route plan would be defined a priori, then building back the schedule based on this knowledge.

The result of this first step is a single service list that could represent a solution of the related *TSP* problem, where a single vehicle should attend all the customers. Trying to start from the solution of the connected *TSP* is a concept that already was be used in several previous solutions (section 3.3.2), but usually, considering that obviously should be adopted a problem simplification in order to admit a single vehicle solution, these heuristics were based on time windows relaxations, considering the geographic proximity as the most important factor when instead temporal constraints were the really compatibility challenge. Our procedure tries to maximize the time constraints compatibility, introducing an aleatory degree necessary to generate at each algorithm execution a new service plan, and basing then the randomness of our solution on this "oracle's prediction".

The next step of the algorithm is the real construction of the routs based on the generated *TSP* solution (line 5) and the publication of the required results (lines 6-7). In the next section will be presented the constructive procedure used to schedule the customer service.

4.2.1 Routes construction

After the definition of the delivery scheduling, the second step is define the subset of clients that each vehicle has to serve. In this section will be presented the simple procedure used to construct the routing plan.

According to the customer order given by the previous phase, each customer is added to the current partial route that minimize the service cost.

$$\begin{cases} f_{k_l} + c_{k_l u} \leq l_u \\ \sum_{i \in R_k \cup \{u\}} q_i \leq Q \\ f_{k_l} + c_{k_l u} + w_u + s_u + c_{u0} \leq l_0 \end{cases} \tag{4.1}$$

Equations 4.1 shows the constraints to allow the insertion of the current customer u into the partial route R_k. The first equation verifies if it's possible to serve the current customer within its time window, where f_{k_l} is the service termination instant of the last customer in the route R_k. The second one checks if the required additional quantity of product required by the current customer is compatible with the total load of the vehicle. The last one it's necessary to check if serving the current customer, the vehicle of the partial route R_k would be able to return to the depot within the maximum available time expressed in the problem definition by the depot due time (see section 2.1.2).

If all these equations are verified, the insertion cost could be computed.

$$cost_k(u) = -c_{k_l 0} + c_{k_l u} + c_{u0} \tag{4.2}$$

$$cost_0(u) = 2c_{u0} + \nu \tag{4.3}$$

Equation 4.2 presents the insertion cost formula used in the algorithm. As consequence of having considered the time constraints in the first phase of the heuristic, in this step the evaluation could be based only on geographical informations, computing the cost simply as the additional service cost introduced by the customer insertion. The cost is then given by the sum of travel distance from the last point of the partial route to the current one and the related returning distance to the depot, subtracting the old return trip cost.

At the same time, the possibility of start a new route has to be evaluated. Equation 4.3 shows the formula used to evaluate the cost of a new vehicle introduction. The round and return travel cost is increased by an additional value ν, representing the necessary cost for the new vehicle usage, in order to disadvan-

tage this alternative, minimizing the routes number. In section 5.1 an exhaustive test on the parameter ν will be presented, empirically showing how the value of this additional cost is not relevant for our purpose, because the algorithm already tends to minimize the number of used vehicles.

Once computed insertion costs for all possible alternatives, the algorithm simply keeps the minimum one proceeding with the customer insertion.

Figure 4.2 graphically represents the route construction procedure.

Analyzing the presented algorithm, the total theoretic complexity of the procedure is $O(n \cdot m)$, recalling, where n is the number of customers and m the number of used vehicles. Comparing it with the *PFIH's* one, that as a time complexity of $O(n^3 \cdot m)$, we obtain a reduction of the running times of 2 degrees of magnitude that is a considerable result taking in account the initial proposal of the research.

Another consideration that could be done on the proposed solution is that, due to the specific design based on time windows constraints, theoretically, the algorithm should better fit on problems with tight time windows, or in general with problems where the time compatibility is the primary challenge in order to find the best solution.

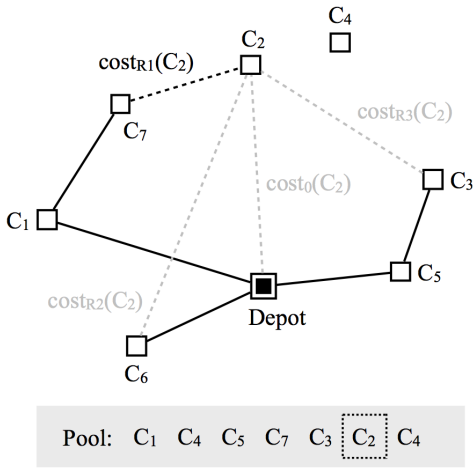
In sections5.3 and 5.4 will be presented a full set of test developed to analyze the results quality and the algorithm performance and to verify the previous theoretics considerations.

4.3 Parallelism

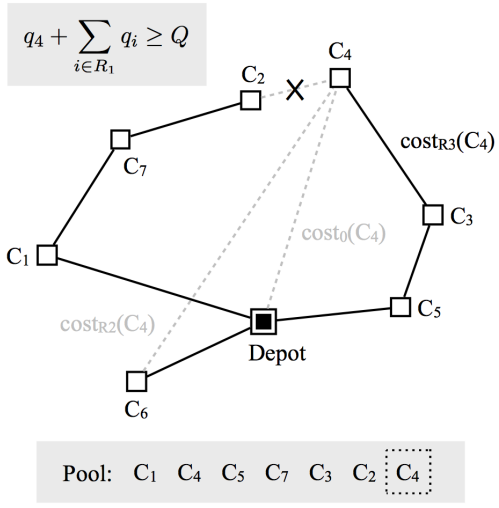
The described algorithm, that realizes the Time Agitation Heuristic, can be implemented following the multi-thread approach.

Considering that each run generates a valid solution and that is theoretically independent from the others, it's possible take advantage of the OpenMP technology lunching several runs of the algorithm in parallel, speeding up the computation.

Once loaded the desired problem instance, it could be used by many threads because it represents a read-only object and all the other support structures used by the *TAH* procedure are initialized directly in its code. Only during the final result update will be required threads synchronization in order to keep the best solution generated.



(a) Insertion cost evaluation, in this figure the route with the minimum cost is chosen



(b) Constraints evaluation for customer insertion, in this figure the capacity constraints for the route with the minimum cost are violated making it ineligible

Figure 4.2: *Time Agitation*: insertion phase. *Source: author*

Experimental test has reveled that the increase in terms of memory consumption isn't a problematic factor and at the same time the required running time is reduced proportionally to the number of available cores.

Chapter 5

Experimental Results

In the following chapter we offer an overview of the experimental results obtained on the implementation of the Time Agitation Heuristic described in the previous chapter, both as independent constructive heuristic and as generator method for genetic algorithm's population. Will be presented a set of experimental tests designed to analyze parameters values, terminations conditions, solutions quality and time complexity, trying to better understand the algorithm behavior and find the best working context.

More in detail, in the first section we will clarify the experimental setups and the used environment in order to better understand the following tests. Parameters configuration and tuning tests are explained in the second section. The third part reports a complete analysis of the obtained quality comparing our results with the literature ones. Aspects regards the randomized structure and the evaluation against deterministic methods will be detailed. Subsequently, the fourth section is dedicated to the complexity analysis, primarily focused on the execution time, and the fifth section presents some considerations about the termination conditions. The last part of the chapter try to evaluate the diversity degree obtained using the *TAH* algorithm to generate the initial population for a genetic algorithm, comparing it with *PFIH*, that is not a naturally randomized procedure, and a complete randomic generator.

5.1 Experimental setup

As already presented in section 2.1.4, in literature are known a large set of benchmark instances for the *VRPTW* problem, used to evaluate and compare the different proposed solutions.

In our work we will use both the Solomon and the Homberger's ones, the first mainly used as reference to better understand the reached quality in terms of solution, and the second ones to compare the time complexity empirically measured. The SINTEF company makes available complete archives with all the original instances definitions on its public website[1][2].

As mentioned in chapter 2, the objective function used for heuristic methods focus primary on used vehicle minimization, and secondly on travel costs minimization.

To understand the real performance of our procedure, being a constructive heuristic, all the experiments are compared with the results obtained by the Solomon's *PFIH* algorithm, already presented at section 3.2, that represents the most used heuristic in that field. The values that will be presented for the *PFIH* are not taken from the literature because many times they are presented as average values of various tests, executed with different configuration, in order to reach the better possible solution. Needing to compare our heuristic against a single run of *PFIH* we have re-implemented the Solomon solution following the exact procedure proposed in his work[43].

We have performed some experiments to understand what will be the *PFIH* configuration that provides the best results and we have observed really small differences between the proposed configurations, then we have chosen to work with its *I1* variants using the $(\mu, \lambda, \alpha_1, \alpha_2) = (1, 2, 1, 0)$ configuration. The obtained results are comparable with the ones presented in other literature.

5.1.1 Environment

The proposed heuristic was implemented in C++, and compiled using *gcc v5.0* for Unix machines. Experiments were run on a Intel Core i7 computer with 2.3 GHz quad-core processor.

[1] https://www.sintef.no/projectweb/top/vrptw/solomon-benchmark/100-customers/
[2] https://www.sintef.no/projectweb/top/vrptw/homberger-benchmark/1000-customers/

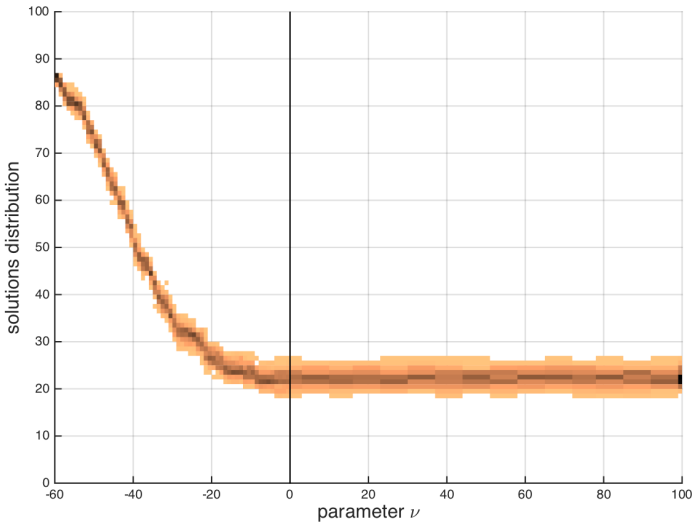


Figure 5.1: Solution distributions changing the ν parameter value on R103 instance. *Source: author*

5.2 Parameters configuration

The Time Agitation Heuristic presented in section 4.2, has a single parameter ν that has to be evaluated and configured before the experiments execution.

Just to remember, the ν constant represents the vehicle cost, that should be the cost that the company has to pay in order to use another vehicle to serve the customers. The introduction of this parameter was justified by the necessity to penalize new tours initialization, trying to maximize the usage of the already started vehicles.

In order to precede with the other experiments, we want to find the better value for ν, on the resolution of the Solomon instances (100 customers). The experiments is then configured as follows.

For each value of ν in the range of $(-60, 100)$, a set of 100 solutions are generated in order to analyze the influence of the vehicle penalty cost in the solution quality.

Figure 5.1 plots the results obtained by the experiment solving the Solomon R103 instance with 100 customers, similar results were obtained for others instances .

For each value of the parameter, is plotted the distribution of the obtained solutions quality, running many times the *TAH* procedure. Darker regions mean higher concentration.

As can be easily seen by the graph, starting from the minimum value, the solution improves with the cost growth until, approximately, the value 0, where starts to be stable until the maximum value. Analyzing all the graphs derived from this experiment, the result is the same, the algorithm reaches a quality maximum at value 0 and then stops to improve.

We can conclude that the algorithm already tends to minimize the number of the used vehicle without the need to introduce a penalizing cost for vehicle insertion. Therefore, proceeding with the other experiments, will be assumed the parameter ν as 0, then simplifying it in the equation 4.3 (section 4.2.1).

5.3 Results analysis

The following section aims to analyze the empirical quality reached by our *TAH* procedure comparing it with others constructive heuristics proposed by literature. Searching for the most used methods, it turns out that many proposed heuristics, like Potvin[38] or Ioannou[22], are in reality an improvement of the original Solomon's one, that means that they use the same structural framework of choosing a first customer to initialize a route and then insert at each iteration the one with the minimum insertion cost. Starting from it, betters formulas are developed in order to improve the computation of insertion costs and first customer election (section 3.2).

Moving from this consideration we have decided to evaluate our algorithm comparing it only with the *PFIH*, that represents the still most used constructive heuristic, quantifying the reached quality as average difference with the best known solution available in literature.

The section will be divided in two main parts, firstly presenting the results obtained solving the Solomon's instance set and then showing the behavior of *TAH* approaching bigger problems like the Homberger's ones.

As already explained in the introduction of this chapter, the *PFIH* results are produced using an our implementation of the Solomon algorithm.

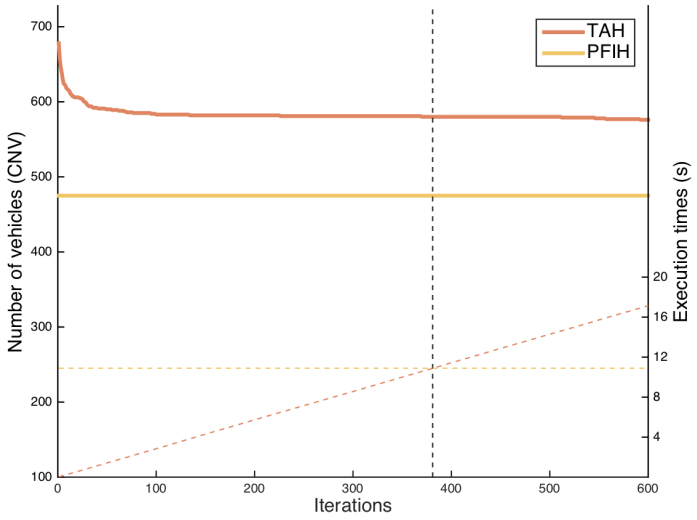


Figure 5.2: *TAH* result rinsing the number of iterations and execution times. *Source: author*

5.3.1 Solomon's instances

In this section will be presented the results obtained solving the Solomon's instances (section 2.1.4). In all the following datas and graphs, if not duly specified, the values refer to average values or cumulative values on all the 56 instances with 100 customers.

In order to compare our randomized algorithm with a deterministic procedure as the Solomon's solution, we have firstly to understand how many times the *TAH* should be run to perform an reliable analysis.

This problem is due to the natural structure of a randomized procedure. As already presented in section 2.3, introducing randomness in a algorithm means try to reduce the computational effort, taking some decision without an exhaustive evaluation of all the alternatives. In this way is expected that the algorithm will be run several times to increase the chances of find a good solution. Here is introduced the problem to estimate when the computation has to stops.

In order to just analyze quality aspects we have decided to proceed with the following tests running our *TAH* procedure until it will be consumed the same computational time needed for the *PFIH*. For this evaluation we don't have used

parallel computing, and the estimation will be computed considering the total time required to solve all the 56 Solomon instances with 100 customers. Time complexity and termination conditions analysis will be presented in the next sections.

In the graph presented in figure 5.2, the solid red and yellow lines represent the results obtained respectively with *TAH* and *PFIH*. Both the values refers to the cumulative number of vehicles (CNV) that is the total number of routes needed to solve all the Solomon's instances. As can be easily seen, the value of *PFIH* is constant because it was performed only one time, and it was plotted for all the graph width to better understand the trend reached by *TAH* rising the number of performed iterations.

The dotted lines are the execution time required by both *TAH* and *PFIH*. Like before, the yellow one, representing *PFIH*, is constant while the *TAH* one represents the total time required by solve all the instances given a certain number of iterations. The interception between these two lines defines the point where the time required to solve one single time all instances with *PIFH* is the same of solving them running x times the *TAH* procedure.

The obtained value is 381 iterations, and in the following analysis all the results will refer to the execution of this number of iterations.

General results

Table 5.1 reports the results, in terms of used vehicle, obtained solving the Solomon's instances as already explained above. In the table are presented the mean results for each of the 6 categories and the cumulative values used in literature to compare results. The second column of the table is the best known solution (BKS) reported in [49]. Then *PFIH's* results are listed paired with the relative percentage error respect to the best solution. The last group lists respectively the *TAH* result, the error respect to the the best solution, and the relative difference with the *PFIH* values.

Figure 5.3 provides a graphical representation of the results, showing the differences between the two constructive heuristics and the comparison with the BKS.

Looking at the resulting datas, can be seen that the *PFIH* provides solutions that are, on average, the 17% higher than the best known solution, while *TAH*'s

Table 5.1: *TAH* and *PFIH* results, in terms of primary vehicle number minimization, on Solomon's 100-customer benchmarks. *Source: author*

Inst	**BKS**	**PFIH**		**Time Agitation**		
		Average	Above	Average	Above	Ab. PFIH
R1	11.92	14.08	18%	17.33	45%	23%
R2	2.73	3.64	33%	4.36	60%	20%
C1	10.00	10.33	3%	11.00	10%	6%
C2	3.00	3.50	17%	4.00	33%	14%
RC1	11.50	14.00	22%	18.75	63%	34%
RC2	3.25	4.13	27%	5.38	66%	30%
CNV	**405**	**475**	**17%**	**580**	**43%**	**22%**

Table 5.2: *TAH* and *PFIH* results, in terms of secondary total travel cost minimization, on Solomon's 100-customer benchmarks. *Source: author*

Inst	**BKS**	**PFIH**		**Time Agitation**		
		Average	Above	Average	Above	Ab. PFIH
R1	1210.33	1147.5	20%	1969.48	63%	36%
R2	951.03	1334.25	40%	1988.64	109%	49%
C1	828.38	997.52	20%	1536.73	86%	54%
C2	589.86	783.99	33%	1283.79	118%	64%
RC1	1384.16	1659.96	20%	2418.05	75%	46%
RC2	1119.24	1663.01	49%	2539.80	127%	53%
CTD	**57187**	**73880**	**29%**	**109272**	**91%**	**48%**

results are 43% higher than the BKS and 22% higher than the *PFIH*.

These results are explained by the higher complexity (as better analyzed in the next section) of the Solomon's algorithm, that tries all possible insertion combinations in order to minimize the costs. Moreover the *PFIH* tends to better minimize the vehicles number because, once started a new route, it proceeds adding customers until possible, while *TAH* adopts a greedy methods that always consider the possibility to start a new route.

As we will better seen in the next paragraphs, for what concerns the generation of population for a genetic algorithms, these results are much closer to the *PFIH*'s ones than could appears from these datas, and the really lower

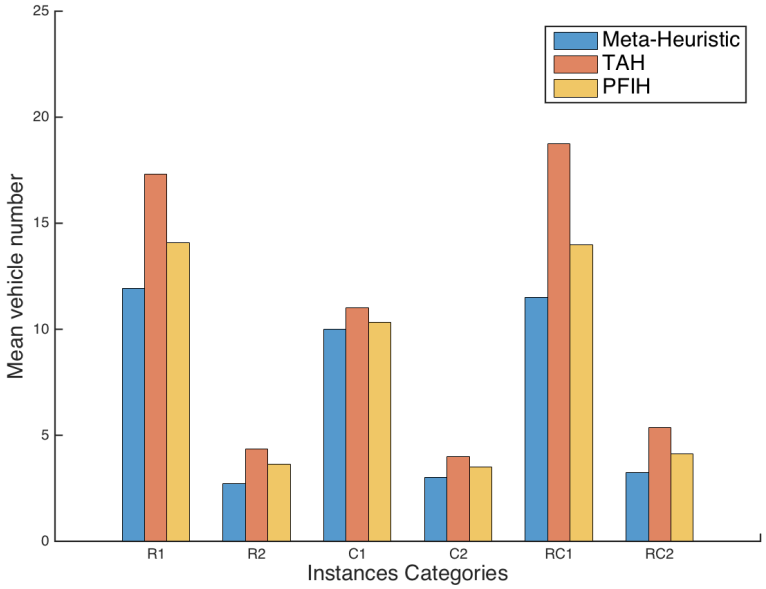


Figure 5.3: *TAH* and *PFIH* results, in terms of primary vehicle number minimization on Solomon's 100-customer benchmarks compared with meta-heuristic BKS. *Source: author*

complexity required by *TAH* justifies the obtained results.

Table 5.2 reports the same informations of the previous one, but showing the values obtained for the total travel costs, the secondary objective function of the heuristic. Looking at the cumulative travel distance (CTD) it's possible to note that the results are worst compared to the CVN values, this fact is caused by the difficulty of a simple constructive heuristic to deal with more minimization functions. The ability to efficiently perform a double minimization function is related to the algorithm complexity, and this justify the lower performance of our *TAH* trying to minimizing even the total travel cost.

Detailed results

In the following paragraph, will be presented a detailed analysis of the results obtained on the Solomon's instances in order to comprehend the behavior of our algorithm respect to each problem class.

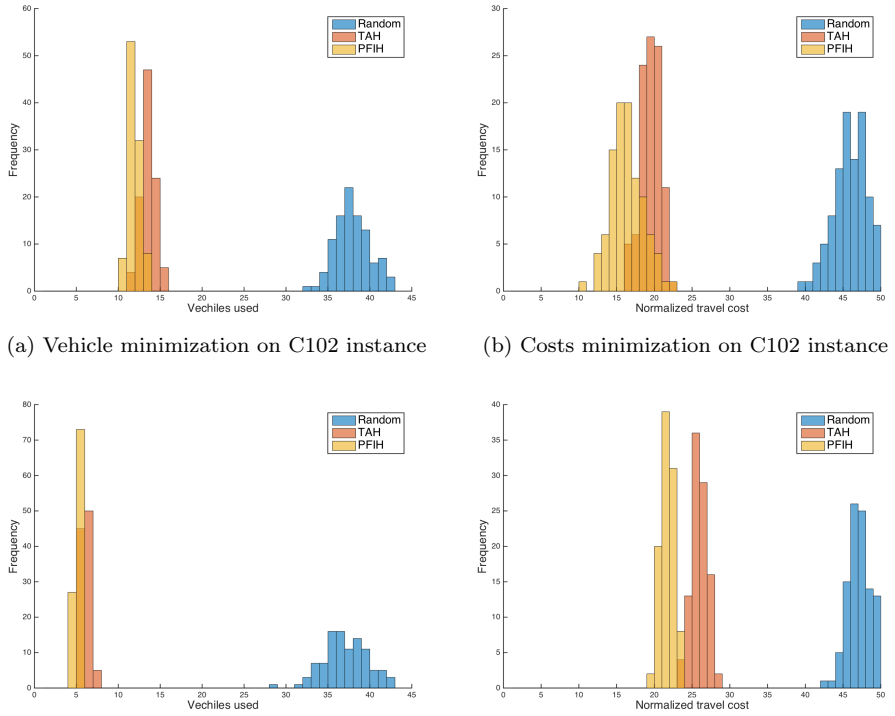
Tables 5.3 and 5.4 shows the detailed results obtained for each single instance

Table 5.3: Detailed *TAH* and *PFIH* results, in terms of primary vehicle number minimization, on Solomon's 100-customer benchmarks. *Source: author*

Inst	**PFIH**	**TAH**	**Inst**	**PFIH**	**TAH**
R101	21	**20**	R201	**5**	**5**
R102	**19**	20	R202	**4**	5
R103	**16**	18	R203	**4**	**4**
R104	**12**	18	R204	**3**	4
R105	**15**	17	R205	**4**	**4**
R106	**14**	16	R206	**3**	4
R107	**12**	15	R207	**3**	4
R108	**11**	17	R208	**3**	4
R109	**14**	17	R209	**4**	5
R110	**12**	17	R210	**4**	**4**
R111	**12**	16	R211	**3**	5
R112	**11**	17			
C101	**10**	**10**	C201	**3**	4
C102	**10**	11	C202	**4**	**4**
C103	**11**	**11**	C203	**4**	**4**
C104	**10**	11	C204	**4**	**4**
C105	**10**	11	C205	**4**	**4**
C106	**11**	12	C206	**3**	4
C107	**10**	11	C207	**3**	4
C108	**10**	11	C208	**3**	4
C109	**11**	**11**			
RC101	**17**	18	RC201	**5**	**5**
RC102	**16**	18	RC202	**4**	5
RC103	**12**	18	RC203	**4**	5
RC104	**12**	19	RC204	**4**	5
RC105	**17**	21	RC205	**5**	6
RC106	**14**	17	RC206	**4**	5
RC107	**13**	19	RC207	**4**	6
RC108	**11**	20	RC208	**3**	6

Table 5.4: Detailed *TAH* and *PFIH* results, in terms of secondary total travel cost minimization, on Solomon's 100-customer benchmarks. *Source: author*

Inst	PFIH	TAH	Inst	PFIH	TAH
R101	1872.80	**1837.94**	R201	**1809.49**	1840.13
R102	**1704.70**	1816.69	R202	**1572.93**	1898.26
R103	**1533.54**	1881.62	R203	**1538.65**	1886.89
R104	**1248.20**	2079.89	R204	**1153.41**	1942.80
R105	**1598.40**	1805.98	R205	**1386.10**	1975.39
R106	**1456.59**	1794.45	R206	**1307.61**	1958.31
R107	**1343.03**	1939.03	R207	**1137.94**	2043.71
R108	**1211.83**	2199.74	R208	**1067.11**	2049.75
R109	**1431.00**	1874.15	R209	**1277.98**	2069.65
R110	**1376.88**	2042.27	R210	**1408.13**	1923.22
R111	**1309.04**	1942.80	R211	**1017.42**	2286.97
R112	**1283.94**	2419.25			
C101	923.71	**828**.94	C201	**603.88**	734.59
C102	**1029.44**	1354.12	C202	**795.61**	1285.53
C103	**1224.39**	1756.29	C203	**934.98**	1731.91
C104	**1150.51**	2033.73	C204	**113403**	1965.08
C105	**880.72**	1161.67	C205	**670.27**	870.87
C106	**952.91**	1335.95	C206	**663.19**	1159.29
C107	**903.91**	1490.67	C207	**749.57**	1182.30
C108	**976.87**	1835.17	C208	**720.36**	1340.74
C109	**935.28**	2034.02			
RC101	**1925.80**	2051.70	RC201	**2077.66**	2274.94
RC102	**1898.41**	2127.82	RC202	**1811.75**	2422.23
RC103	**1545.31**	2394.46	RC203	**1534.89**	2434.37
RC104	**1372.14**	2737.36	RC204	**1257.53**	2486.59
RC105	**1858.66**	2344.01	RC205	**2147.25**	2398.10
RC106	**1730.54**	2297.09	RC206	**1746.84**	2604.13
RC107	**1577.04**	2526.89	RC207	**1569.26**	2658.45
RC108	**1371.76**	2865.07	RC208	**1158.86**	3039.57



(a) Vehicle minimization on C102 instance

(b) Costs minimization on C102 instance

(c) Vehicle minimization on R201 instance

(d) Costs minimization on R201 instance

Figure 5.4: Population generation for Solomon's 100-customer benchmarks. *Source: author*

respectively for the primary vehicle number minimization and for the secondary total cost minimization.

From tables comes out that *TAH* produce, in terms of vehicle minimization, a better solution in one case, reaches the same results in other 12 instances, and for other 23 instances the result produced differs for a single vehicle respect to the *PFIH* solutions, therefore being a competitive method on 36 of the 56 problems.

Population generation results

In order to better understand the behavior of our *TAH* algorithm when used to initialize the population of a genetic algorithm, we have analyzed the distributions of the generated individuals produced by *TAH*, a randomized version of

PFIH and a complete randomic generator.

Being a complete deterministic procedure, the pure *PFIH* as presented by Solomon couldn't be used to generate a genetic population because it computes always the same result. To use it in that context, many authors[1, 14, 24] have used a version called Stochastic *PFIH* that introduces a randomness degree simply choosing the first customer at random, instead using an estimation formula. For this experiment we have implemented a stochastic *PFIH* using the same insertion configuration explained in the previous sections (5.1).

The complete randomic generator was implemented generating a random permutation of the customer ids. A string composed by any permutation of ids could be used as chromosome representation in GA (codification *GR3* defined by Yi-Liang et Al.[50]). In this case the codification in given by the simple juxtaposition of the string representing the single routes and then the optimal underlying solution can be constructed using a split function, like the one proposed by Prins in [39], that is able to define the best string partitioning to re-construct a valid solution (section 3.3.2).

In our experiment we have generated a population of 100 individuals for each proposed generator. Figure 5.4 graphically shows the distributions resulting from the experiment on two sample instances.

Graphs 5.4a and 5.4c reports the distributions respect to the vehicle minimization, while 5.4b and 5.4d are about total travel costs. In all the graphs the abscissa represents the reached quality, the two graphs about the secondary minimization were normalized in order to better visualize the results and obtain a qualitative point of view.

Looking at the images, that show an entire population pool instead the only best solution, is more clear the quality of the proposed heuristic, which solutions fall in the same range of PFIH's ones for almost the majority of the cases.

5.3.2 Gehring and Homberger's instances

Like already done with the Solomon instances, in the following section will be presented the analysis of results obtained solving the Homberger instance set (see 2.1.4).

The entire structure of the performed experiments follows the structure explained above. In this case, the number of *TAH* iterations that equalizes the total execution time to solve the 300 instances is 945. A complete analysis about

Table 5.5: *PFIH (a)* and *TAH (b)* results, in terms of primary vehicle number minimization, on Gehring and Homberger's benchmarks with 200 to 1000 customers. *Source: author*

Inst	**PFIH results**					
	200	**400**	**600**	**800**	**1000**	**Total**
R1	20.4	39.6	60.8	80.2	102.1	303.1
R2	4.8	8.3	11.7	15.5	19.2	59.5
C1	20.5	40.1	59.9	79.8	99.3	299.6
C2	7.0	13.5	20.7	27.0	34.2	102.4
RC1	19.4	38.3	56.6	77.2	92.6	284.1
RC2	5.4	10.2	13.9	18.6	21.1	69.2
CNV	**775**	**1500**	**2236**	**2983**	**3685**	**11179**

Inst	**TAH results**					
	200	**400**	**600**	**800**	**1000**	**Total**
R1	18.9	38.0	59.3	76.7	100.0	292.9
R2	5.2	9.4	12.9	15.6	19.5	62.6
C1	22.5	46.4	73.9	101.1	131.5	375.4
C2	7.9	15.8	25.0	36.0	47.3	132.0
RC1	21.5	45.7	64.3	83.8	102.4	317.7
RC2	7.5	16.2	22.8	29.6	31.8	107.9
CNV	**835**	**1715**	**2582**	**3428**	**4325**	**12885**
Above	7.7%	14.3%	15.5%	14.9%	17.4%	15.3%

time complexity will be presented in the next section.

Tables 5.5a and 5.5b show the results obtained minimizing the vehicle number. All the values are computed in the same way of the previous case, reporting the mean value for each class and the cumulative value for the complete set.

How can be seen looking at the percentage error between the two constructive heuristics, *TAH* reaches results that are, in average, 15% higher than the Solomon algorithm. This is a better result compared with the Solomon instance set and means that the proposed heuristic work well with big-sized problem like the Homberger's ones.

More in details we could observe that for the R1 class our heuristic steadily

Table 5.6: Execution times on Solomon's 100-customers benchmark compared with a random generation and the *PFIH* algorithm (values in milliseconds). *Source: author*

Inst	**Random**	**TAH**	**PFIH**
R1	9.82	7.42	783.24
R2	14.01	4.36	4397.43
C1	7.30	5.42	716.69
C2	7.38	3.11	2275.49
RC1	5.94	4.79	490.74
RC2	8.70	3.47	2594.09
Total	**53.15**	**28.57**	**11257.68**

obtains better results, that, in average, reduce the number of used vehicle by 3.4%, and in R2 class the results are absolutely competitive, generating solutions only 4.5% higher than the *PFIH*'s ones.

How seen in previous experiments, *TAH* generate solutions in average 15-20% higher than *PFIH*. How will be shown in the next section, this difference is justified by the really lower complexity of the algorithm, that is able to compute solution for even big-sized problem within a really short time.

5.4 Complexity analysis

After the analysis of the quality obtained by our *TAH*, in this section will be presented a complete evaluation of the algorithm complexity, comparing the time execution of the various algorithms and trying to extract the empirical complexity. We will concentrate on time complexity analysis because in this case the spatial complexity doesn't represent a constraint.

In order to better understand the *TAH* performance, the following experiments aim to measure the time required by a single iteration of the algorithm, because given the behavior of a single run, the time required to execute more iterations is computable simply multiplying the single time. Considerations about the execution of multiple iterations will be presented at the end of this section and in the next one.

In the next experiments will be presented the time required to produce complete randomized solutions, too. We have decided to include that comparison

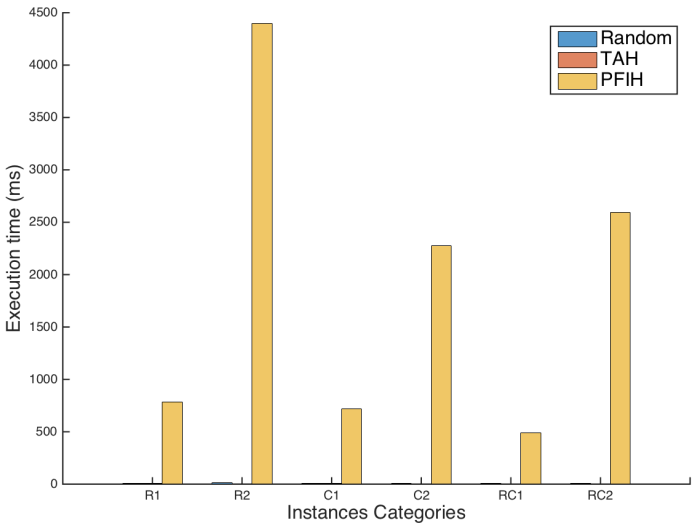


Figure 5.5: Execution times on Solomon's 100-customers benchmark compared with a random generation and the *PFIH* algorithm. *Source: author*

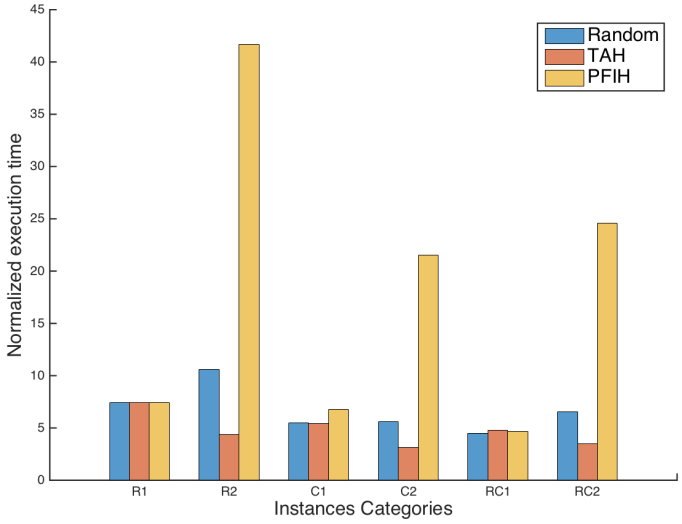


Figure 5.6: Normalized execution times on Solomon's 100-customers benchmark compared with a random generation and the *PFIH* algorithm. *Source: author*

because, theoretically, it should represent the simplest method to compose a valid solution defining a lower bound for the computational time and then providing a good metric to evaluate other heuristics. The algorithm adopted to generate this randomized solution is the one already presented in section 5.3.1 and widely used in literature to initialize a genetic algorithm population. For the *PFIH* algorithm was used the classical deterministic variant proposed by the original Solomon's paper.

Table 5.6 reports the total time required by the three methods to solve, one time, all the 56 instances of the 100-customers Solomon's set (figure 5.5 provides also a graphical representation). The first consideration that could be done looking at these datas, and that could sound strange, is that the time required to solve the problems using the *TAH* is even lower than using the random procedure. This fact is justified by the structure of the randomic generator algorithm. As explained in the previous section, after the computation of a customer's ids permutation, to produce a valid solution it's required a split function like the Prins one.

The split function computes the cost of all the possibles feasible routes starting from the predefined order, then each route is added as a node in a graph representation connecting two nodes if the relatives underlying routes are adjacent substrings in the original permutation. To produce a valid scheduling is then performed a shortest path algorithm on the so defined graph to reach the last customer starting from the depot node. This briefly described algorithm guarantees the definition of the best scheduling plan given a predefined permutation of the customers (for more information see section 3.3.2).

The higher complexity of this randomic generator compared to the *TAH*, is due to the fact that the graph construction procedure and the short path algorithm are themselves more complex than our procedure, and this justify the experimental results.

After verifying that the *TAH* has a time complexity completely comparable with the randomic generator, let's see the comparison with the *PFIH* results.

How can be easily seen by the figure 5.5, the time required by the Solomon algorithm is some degree of magnitude higher than the *TAH*, precisely 3 for the results on this set.

The *PFIH* is able to reach better solution analyzing each time the best insertion step and this means test all possible insertion positions in all the partial

Table 5.7: Execution times on Homberger benchmark compared with a random generation and the *PFIH* algorithm (values in milliseconds). *Source: author*

Inst	**Random**	**TAH**	**PFIH**
200	$1.03 \cdot 10^2$	$9.47 \cdot 10^1$	$6.84 \cdot 10^4$
400	$1.99 \cdot 10^2$	$3.38 \cdot 10^2$	$2.84 \cdot 10^5$
600	$2.91 \cdot 10^2$	$7.04 \cdot 10^2$	$6.31 \cdot 10^5$
800	$3.73 \cdot 10^2$	$1.19 \cdot 10^3$	$1.15 \cdot 10^6$
1000	$4.58 \cdot 10^2$	$1.78 \cdot 10^3$	$1.76 \cdot 10^6$
Total	$\mathbf{1.43 \cdot 10^3}$	$\mathbf{4.12 \cdot 10^3}$	$\mathbf{3.89 \cdot 10^6}$

routes. How already explained in section 4.2.1 this fact leads to a theoretical time complexity of $O(n^3)$ paying the quality with a time factor of 390x to generate the solution.

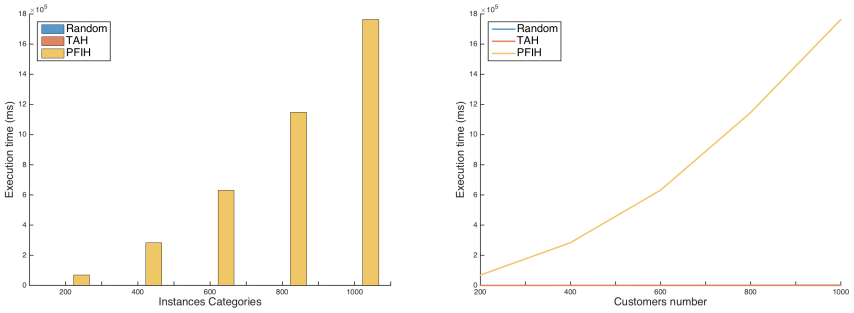
Considering that this difference is obtained on a relatively small problem (100 customers), the high time complexity of *PFIH* should be taken in account when it's necessary to work in some environment like real-time applications or high loaded online services. Depending on the context requirements, the *TAH* could represents a good choice for the initialization of a genetic algorithm population, too, providing solution really closed to the *PFIH* ones but within a time interval comparable with a completely randomic generation.

Time complexity sensibility

Figure 5.6 shows the same execution time of the previous graph but normalized in order to highlight eventual differences between classes. To normalize the values, all the times of the *PFIH* and random executions are divided by a factor computed by equalizing the three values for the R1 class.

Looking at the histogram it could be seen how *TAH* and *PFIH* have different behaviors respect problem class 1 and class 2. The *PFIH*, for each of the three type (R, C, RC), obtains results that are significantly higher for class 2 than for class 1, while *TAH* shows a complete opposite behavior.

How already presented in section 2.1.4, classes 1 has a short scheduling horizon and this mean that the solution tends to be composed by many vehicles that are able to serve only a few customers, while classes 2, with long scheduling horizon, induces plans with few vehicles, each one serving many customers.



(a) Execution times per instance class

(b) Execution times growth increasing the number of customers

Figure 5.7: Execution times on Homberger benchmark compared with a random generation and the *PFIH* algorithm. *Source: author*

Therefore, how could be theoretically predictable, *PFIH*'s complexity is more sensible to the mean route size, requiring a lot of time to analyze each possible insertion point in routes, instead *TAH* is more sensible to the routes number, having to analyze each vehicle in order to choose which one would minimize the service cost. Even thought will not be explicitly presented in that work, the experiment conduced on the Homberger's instances confirms the just presented result.

Time complexity order analysis

In the following part, we will analyze the growth of the execution time increasing the number of customers that should be served. To study this behavior it was be necessary realize an experiment using the Gehring and Homberger's instances, which, providing problems starting from 200 customers to 1000, allow a better analysis than the small Solomon's ones.

In the same way as already done with the previous instances, table 5.7 shows times required by the three methods to complete the single resolution of all the 300 instances of the set. To facilitate the reading and underline the order of magnitude, all the values were reported in scientific notation.

The behavior of the time complexity is coherent with the previous considerations. As could be easily seen, *PFIH* constantly requires times with three degrees of magnitude more than the *TAH*, increasing so quickly that with the

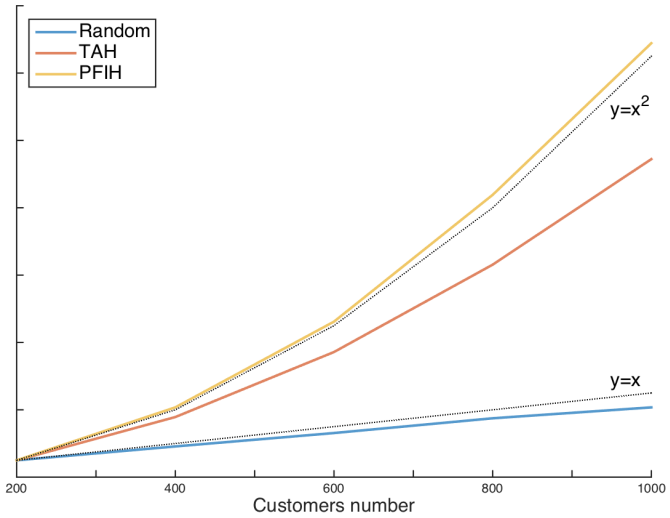


Figure 5.8: Heuristic's time complexity analysis. *Source: author*

1000-customers problem, it results 985 times slower than our algorithm, and 3790 times slower than the randomic generation.

A brief observation about the comparison between randomic procedure and *TAH* should be done. In the previous experiments, on Solomon's instances, we have noted times always lower for our algorithm, and this was justified by the structure of the randomic generator. As can be seen by these new datas, until 200 customers, *TAH* still performs better results, but increasing more the customer number it turns slower. This fact doesn't invalidate our previous analysis, because the higher values obtained with "small" instances is due to some constant costs to generate auxiliary structures and solve the shortest path algorithm that don't grow linearly with the number of customers requiring lower times solving bigger problems.

Due to so high constant factors, looking at figure 5.7, that shows the obtained curves for the observed complexity, it's impossible to understand the real difference between algorithms asymptotic behaviors.

In order to better understand how the algorithms scale increasing the number of customers, figure 5.8 reports the execution times normalized and reduced by relative multiplication factor constants.

The graph in figure shows the asymptotic behaviors comparing them with

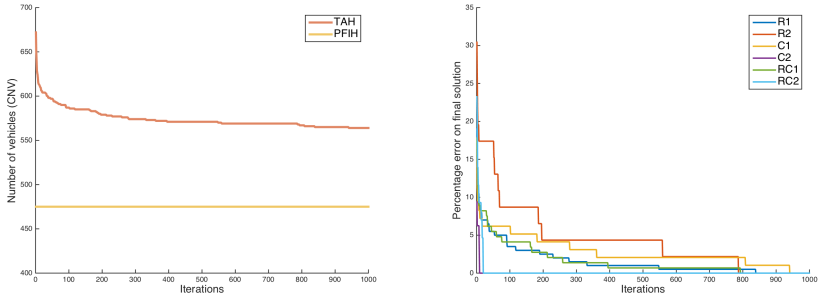
Table 5.8: Results trend increasing the number of iterations on Solomon's 100-customers benchmark, times values in milliseconds, classes values expressed like mean value. *Source: author*

	Time	**CNV**	**R1**	**R2**	**C1**	**C2**	**RC1**	**RC2**
PFIH	11257	**475**	14.0	3.6	10.3	3.5	14.0	4.1
TAH(1)	28	**672**	19.8	5.5	12.7	4.3	21.5	6.6
TAH(10)	281	**613**	18.1	4.9	11.6	4.0	19.9	5.9
TAH(50)	1402	**597**	17.6	4.9	11.4	4.0	19.3	5.4
TAH(100)	2800	**587**	17.3	4.6	11.4	4.0	19.0	5.4
TAH(394)	11246	**572**	16.8	4.4	11.0	4.0	18.5	5.4
TAH(1000)	28572	**564**	16.6	4.2	10.8	4.0	18.3	5.4

the linear and quadratic functions. From the empirical analysis it's possible to see that the randomic generation actually follows a linear behavior. Looking at the *PFIH*'s curve, theoretically it should be expected a $O(x^3)$ behavior while the empirical result shows a lower $O(x^2)$ complexity. This difference between theoretical and real values is due to the fact the the big-O notation represents always the worst-case value, but thinking at the *PFIH* procedure, at the beginning there are many possible customers to test in a really small number of possible insertion points and continuing the algorithm the number of insertions position rises but the number of customers to test will be always smaller, requiring a time lower than the expected. This consideration justifies the empirical *PFIH* time complexity.

Despite the difference between *TAH* and *PFIH* asymptotic behaviors is not so high as expected, resulting only a little lower than the Solomon's algorithm for infinite asymptotic, working on problems that have sizes like the ones from instances, we can't simply drop out the multiplication factors in order to estimate the time required to solve a problem, because this would provide a false result.

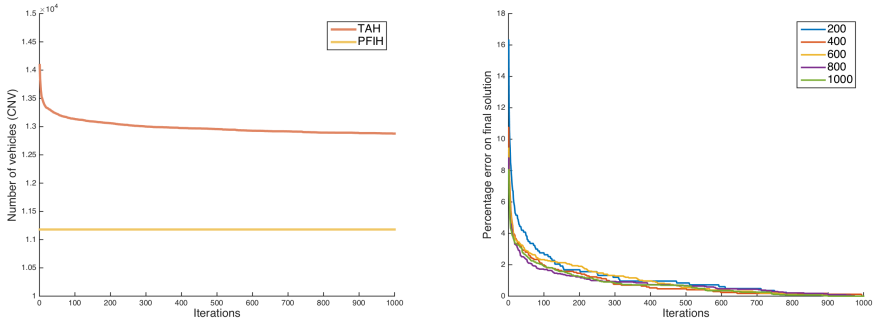
Dealing with factors that are in the same order of the problem domain, means in practice increase the real complexity, so we can assert that solving the instances coming from the benchmark sets, the observed complexity of *PFIH* is a $O(x^3)$ against the *TAH* of $O(x^{1.8})$, turning the *TAH* an interesting solution for certain application fields.



(a) Obtained CNV compared with *PFIH* solution

(b) Relative error on final solution trend for classes

Figure 5.9: Trends increasing iterations on Solomon benchmark. *Source: author*



(a) Obtained CNV compared with PFIH solution

(b) Relative error on final solution trend for size classes

Figure 5.10: Trends increasing iterations on Homberger's set. *Source: author*

5.5 Convergence analysis

The following section is dedicated to the evaluation of the algorithm improvement increasing the number of iteration executed, and to the possible termination conditions that could be used with an heuristic like the one proposed.

Table 5.8 reports the CNV values, and the mean number of vehicle for each problem class, obtained for different number of performed iteration on Solomon's instances. Figures 5.9a and 5.9b show the values presented in the table in order to better understand the general behavior (values of 5.9b represent the relative percentage error on the best result obtained with the maximum number of iterations).

Table 5.9: Results trend increasing the number of iterations on Gehring and Homberger's benchmark, times values in milliseconds, classes values expressed like sub-total values. *Source: author*

	Time	**CNV**	**200**	**400**	**600**	**800**	**1000**
PFIH	$3.89 \cdot 10^6$	**11179**	775	1500	2236	2983	3685
TAH(1)	$4.15 \cdot 10^3$	**14089**	970	1897	2823	3729	4673
TAH(10)	$4.12 \cdot 10^4$	**13453**	894	1795	2699	3570	4495
TAH(50)	$2.06 \cdot 10^5$	**13224**	866	1761	2655	3505	4437
TAH(100)	$4.12 \cdot 10^5$	**13134**	857	1747	2640	3483	4407
TAH(394)	$3.89 \cdot 10^6$	**12885**	835	1715	2582	3428	4325
TAH(1000)	$4.12 \cdot 10^6$	**12877**	834	1713	2580	3425	4325

First at all we can note that there are not big differences between classes, that have a comparable convergence. The obtained curves shows that the algorithm has a early really rapid improvement phase that decrease after a few iterations stabilizing the solution.

These results can be confirmed looking at the same kind of data computed on testing the Gehring and Homberger's instances (table 5.9 and figure 5.10). In this case the datas were divided into problem-size classes (this time the values in the table represent the partial total number of vehicles).

As for all the randomized algorithms, the termination conditions depends on the application area, representing a trade-off choice between final result quality and running time, then is not possible provide a certain estimation for the "best" number of iterations.

However, should be taken into account that the execution of several iterations is not necessary a sequential procedure, because each single solution is completely independent to others and could be processed in parallel. As already demonstrated in similar works[2], this kind of randomization scales linearly on a multi-threads processor, providing a perfect parallelism and reducing the computation times by a factor that is equal to the number of physical cores available.

Unfortunately for this work we don't have access to a many-core machine to realize good experiments on parallelization performance, but we have reasons to believe that this approach works well on *TAH*, too.

Considering the parallelism exploitation we obtain that *TAH* is an algorithm that improves simply with the evolution of the technology, since with a future

many-core processor would be possible to run thousands of iteration simultaneously providing a better solution within the execution time of a single iteration.

5.6 Diversity control analysis

Desiring to develop a new constructive heuristic that would be used in genetic algorithms context for the population initialization, an important factor will be the ability to generate dispersed individuals in order to maximize the pool diversity. As already mentioned in chapter 2 (section 2.4.1), the diversity control of generations is a key factor in order to prevent premature convergence and encourage a wide analysis of the problem domain.

In the present section will be presented the results of an experiment designed to measure the diversity property, firstly presenting the metric functions used to compute an estimation of the property and then reporting the values obtained analyzing in that way the populations generated with a random generator (see 5.3), the *TAH* and the Stochastic *PFIH*.

Before passing to the analysis of the distance functions, we have to define the representation of a *VRPTW* solution. In section 5.3.1 we have already introduced the codification *GR3* defined by Yi-Liang et Al.[50], where the vectors representing the customers served by each vehicle are simply juxtaposed generating a generic permutation of the customer ids without an explicit representation of the routes delimiters. Based on this consideration, we propose two different formulas to compute the distance between two given solution.

The first estimator is based on the method introduced by Vidal et al. [47] to define a population diversity metric, where the contribution of a individual to the total diversity was measured like the mean distance between it and its neighbors, using the simple Hamming distance to compute the difference between the representations of two solutions.

In order to calculate the diversity property Δ_{pop} of the whole population we have extended the previous definition measuring the mean distance between all the possible pairs of individuals in the given pool.

$$\begin{aligned} \Delta_{pop} &= \frac{1}{n} \sum_{P \in pop} \Delta(P) \\ &= \frac{1}{n(n-1)} \sum_{P \in pop} \sum_{P2 \in pop} \delta(P, P2) \end{aligned} \tag{5.1}$$

Equation 5.1 shows the used formula, where δ is the distance function. Given the previous representation of an individual, δ can be defined using any known string metric that measures the differences between two sequences. Between all the possibilities, our choice was the Levenshtein function[29], that counts the minimum number of single-character edits (insertion, deletions or substitutions) required to change one word into the other. In our opinion, this is the best way to evaluate the differences between two permutations.

$$\delta_L(a,b) = lev_{a,b}(|a|,|b|)$$

$$lev_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

The second estimator is based on the idea that it's possible to enumerate all the permutations of a given string, defining a total order that allows computation of the string position respect to this domain.

Given a string S composed by m different characters, the number of possible permutations of the m chars is $m!$. A total order function on the characters then induces a total order on the strings domain, simply using an alphabetical sorting. Moreover, knowing that all the possible strings have the same length and that could not be repeated any character in them, it's possible compute the absolute position of a given string respect to the whole domain. In order to simplify the notation, in the following formulas it's assumed that the chars are represented by integer numbers, where then the values itself are the absolute positions in the characters domain.

$$\psi(\mathrm{S}) = \sum_{i=1}^{m} \left[\left(\mathrm{S}[i] - 1 - \sum_{j=1}^{i-1} \mathbf{1}\left(\mathrm{S}[j] < \mathrm{S}[i]\right) \right) \cdot (m-1)! \right] \tag{5.2}$$

Equation 5.2 shows the formula for the computation of the absolute position of a string in the relative domain. The notation $\mathrm{S}[i]$ obviously indicate the i-th character of the string and the expression $\mathbf{1}(cond)$ has value 1 if and only if the condition is verified and 0 otherwise.

$$\gamma(S_1, S_2) = |\psi(S_1) - \psi(S_2)| \tag{5.3}$$

Table 5.10: Population diversity computed with Delta and Gamma estimators on Solomon's 100-customers instances. *Source: author* .

Inst	**Δ_{pop} Estimator**			**Γ_{pop} Estimator** ($\times 10^{156}$)		
	Random	**TAH**	**PFIH**	**Random**	**TAH**	**PFIH**
R1	98.06	94.07	93.23	1.231	1.061	0.864
R2	98.06	96.35	86.97	1.231	1.124	0.902
C1	98.08	93.25	90.53	1.222	1.062	0.913
C2	98.06	91.30	70.83	1.240	1.027	0.833
RC1	98.06	95.55	93.68	1.231	0.934	0.950
RC2	98.07	96.25	86.90	1.253	1.098	0.843
Total	**98.06**	**94.46**	**87.03**	**1.234**	**1.051**	**0.884**
Diff		3.7%	11.2%		14.8%	28.4%

After the definition of the absolute position, the distance between two strings is simply calculated as the difference of the two indexes (equation 5.3).

Using the previous definitions, the diversity estimator Γ_{pop} of a population, evaluated with this second method, will be computed like the mean distance between two consecutive individuals after the alphabetical sorting of the population (equation 5.4), in this way an higher value of gamma means a wider exploration of the problem domain.

$$\Gamma_{pop} = \frac{1}{n-1} \sum_{i=1}^{n-1} \gamma(\text{pop}[i], \text{pop}[i+1]) \tag{5.4}$$

Table 5.10 shows the diversity values obtained using Delta and Gamma estimator on the 100-customers Solomon's instances.

Considering that the random generator structurally provides the better solution in terms of population diversity, other heuristics has to be compared against it trying to maximize the estimators values.

How could be seen, *TAH* generates a better population, reaching diversity degrees constantly higher than the stochastic version of *PFIH*. This result could be justified by the structural randomness of the *TAH*.

Indeed, while *TAH* already uses randomization in order to solve the *VRPTW* problem, the *PFIH* is naturally a complete deterministic procedure, designed to generate always the same result. The stochastic version should introduce a

randomness degree in the algorithm, but the structure of the solving procedure tends to converge always to the same result, because, even if the first customer of a route is chosen randomly, probably the insertion compatibility evaluated in the second stage would tends to reconstructs the same route, prescinding then from the first choose.

We can conclude asserting that the TAH offers a better generator for $VRPTW$ genetic algorithm in terms of diversity control, providing solution with a quality comparable with the $PFIH$ solver, but with times and randomness comparable with the randomic generation.

Chapter 6

Conclusions

In this chapter will be summarized the presented work, commenting the experimental results given in the previous chapter.

As introduced in the first chapter, our initial intent was the introduction of new heuristic able to find a acceptable solution in a really short time. The trade-off between performance and quality always is the main challenge developing new algorithms, and usually the best way to understand the usefulness of a solution is not evaluate it in absolutes terms, analyzing only time performance or reached quality. To understand if a solution represents a competitive alternative in some applications, it could be introduced the concept of Pareto-optimal.

The idea is to imaging a graph where the two dimensions represent a quality metric and a time performance. Plotting in that graph all the existing methods for a certain problem, one solution is called Pareto-optimal if there is not other solutions that are better both for quality and performance.

In that way the subset of Pareto-optimal solutions represents the best state-of-art providing methods that satisfy different necessities, because depending on application requirement a worst solution could be preferable if the execution time should be really short.

With the results obtained analyzing our Time Agitation Heuristic, we believe to have defined a new Pareto-optimal alternative, providing an acceptable solution within a time comparable with random generators. Moreover, the exploitation of the randomic approach allows the improvement of the resulting quality without affecting in any way the performance, simply running the algo-

rithm in a parallel environment. This could be a good achievement considering the current technology tendency, where the number of cores in a processor tends to considerably increase and the mobile application requires always faster algorithm to provide real-time services.

Developing the presented heuristic, we have tested several more complex variants. We have try to introduce insertion costs weights in order to encourage the return to the depot for a high loaded truck, trying to reproduce a more natural behavior. Then we have tested a look-ahead strategy to evaluate more than one client per time trying to preventing forced insertion that increase the total service cost. And finally we have tested a complete different approach generating a compatibility graph between clients and then exploiting a really new powerful algorithm for the clique covering to minimize the number of required vehicles.

All these attempts have proven to works correctly reaching valid solutions, sometimes representing a really original theoretical approach, but everyone has provided worst results in terms of quality or time performance, turning the simple presented heuristic the best result obtained by the research.

Another interesting aspect came out from the analysis, is the good behavior in terms of diversity control. As already mentioned in the previous chapters, the premature convergence is a really important aspect in genetic algorithm's design, and many times the complete randomic generation is the suitable method, preferring to prevent local minima than initialize an already good population.

In this context our algorithm could provide a good compromise, generating pseudo-randomic solutions, based on time window constraints, with a good quality without affecting the necessary diversity of the individuals.

The last aspect that we want present is the introduction of a new methodology instead of a single punctual solution. As already reported in the previous chapters, all the existents constructive heuristics, or at least all the used ones, are based on the *PFIH* framework, improving estimation formulas but still using the insertion process proposed by Solomon in 1987. Our heuristic explore a complete new approach, basing it structure on different concepts.

The idea to solve the time constrained *VRP*s starting from a randomly chosen instant in the customer time windows, could be used not only to solve the specific *VRPTW* variant, but all the classes of routing problems that have to satisfy time limitations.

Chapter 7

Future developments

In this chapter we would propose some possible future developments both to continue the analysis of the proposed heuristic, evaluating it's impact on real application, and to extend it, improving quality and performance results. In the last part of this chapter will be presented a possible new algorithm based on the proposed framework that tries to solve the *VRP* problem using a completely new approach.

As already mentioned, our *TAH* algorithm was developed to be used inside a genetic algorithm as constructive heuristic to initialize the population. For that reason, when the complete genetic algorithm will be available, it should be tested the real influence of this initialization procedure on the final result. Sensitivity tests should be performed to analyze how much a good initial solution impacts on the genetic algorithm and to compare the differences between the usage of *TAH* instead of stochastic *PFIH* or randomic generators.

In the previous chapter we have remarked that our algorithm introduces a new resolution methodology that could be used to solve all the *VRP* variants with time constraints. For this reason could be used the *TAH* to solve the multiple depots problem (*MDVRPTW*), where, as suggested by the name, a vehicle could depart from various depots. Our algorithm could simply adapted for this generalization just considering various possible new routes instead of only one. In this way, when a new vehicle is required, it would be allocated on the nearest warehouse. In this case we suggest to implement a look-ahead strategy in order to prevent vehicle allocations based on a single customer position.

During our research, we tested several variants of the *TAH*, among that look-ahead strategies and dynamic vehicle choice probabilities based on load and position. All these solutions have reported little solution improvements that don't justify the higher computational complexity. Nonetheless we believe that in some cases, with a better configuration of the variants and a deeper understanding of the statistical implications, these approaches could bring to better solutions. Here we will briefly introduce a *TAH* variant, called *TAH-*R^2Q, that approaches *VRPTW* problems with a completely new prospective and that we believe could improve current results working on a better algorithm configuration.

One of the most interesting aspects proposed with the *TAH* is the a-priori arrival time estimation that exploits the randomic approach to generate a stochastic forecasted scenario, where the *VRPTW* is reduced to a simpler scheduling problem with specific service times.

Having a precise required arrival time at customers, considering two points we can previously know if they could be served by a same vehicle, and then if they are each other compatibles. Leaving aside for a moment the capacities constraints, starting from this consideration, we can define a compatibility graph where each node is a customer and each edge represents a service compatibility. Solving a clique covering problem on the defined graph, we obtain the optimal routing plan because each clique represents a vehicle and the algorithm naturally minimizes the number of required cliques.

Therefore, the *TAH-*R^2Q algorithm translates a vehicle routing problem in a clique covering problem, still a NP-hard problem that haven't really good solutions until now. In 2014, Badini, Bodini and Cattani[2] introduced a completely new randomic clique covering algorithm, based on Karger's min-cut algorithm. Applying it to hardware high-level synthesis, it has proven to represent a really good solution providing great improvements respect to existing exact solutions.

A first implementation of the presented *TAH* variant doesn't provide good solutions, both in terms of quality and time. Due to the good results provided by the clique covering algorithm on other problems, we think that a deeper analysis on configurations and compatibility definition could significantly improve the results. In our opinion the first step should be statistically analyze the arrival times for already optimally solved solutions in order to better understand the real distribution of these instants in the customer time windows. When a bet-

ter empirical probability distribution will be defined the stochastic forecasted scenarios should better fit with real solutions, then generating a more realistic compatibility graph.

Another possible improvement could be the exploitation of unfeasible solutions introducing penalized compatibility arcs that violate some constraints. As already presented in the previous chapters this approach allows a better exploration of the high quality solution sub-domain.

In terms of performance, we have observed that the high time complexity required by this version is due to the utilization of the Boost libraries[1] to manipulate graphs and perform the clique covering. In our implementation, was used the same code developed for the hardware synthesis optimization, that was designed to be integrated in a more complex environment. We believe that an ad-hoc implementation of this algorithm optimized for the $VRPTW$ could significantly reduce the computational time.

Moreover the double-randomized approach allows real low level parallelization, this characteristic turns the algorithm a good candidate for a parallel version exploiting the modern many-cores GP-GPUs.

[1] http://www.boost.org

Bibliography

[1] Guilherme Bastos Alvarenga, Geraldo Robson Mateus, and G De Tomi. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. *Computers & Operations Research*, 34(6):1561–1584, 2007.

[2] Federico Badini and Stefano Bodini. A randomized algorithm for functional unit sharing. In *Master Degree Thesis*, pages 87–88. Politecnico di Milano, 2015.

[3] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5):1269–1283, 2011.

[4] J Ben Atkinson. A greedy look-ahead heuristic for combinatorial optimization: An application to vehicle scheduling with time windows. *Journal of the Operational Research Society*, pages 673–684, 1994.

[5] Russell Bent and Pascal Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

[6] Olli Bräysy. A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS Journal on Computing*, 15(4):347–368, 2003.

[7] Olli Bräysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation science*, 39(1):104–118, 2005.

[8] Yaw Chang and Lin Chen. Solve the vehicle routing problem with time windows via a genetic algorithm. *Discrete and Continuous Dynamical System Supplement 2007*, pages 240–249, 2007.

[9] Jean-François Cordeau and Québec) Groupe d'études et de recherche en analyse des décisions (Montréal. *The VRP with time windows*. Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[10] Jean-François Cordeau, Gilbert Laporte, and Anne Mercier. Improved tabu search algorithm for the handling of route duration constraints in vehicle routing problems with time windows. *Journal of the Operational Research Society*, 55(5):542–546, 2004.

[11] Jean-François Cordeau and Mirko Maischberger. A parallel iterated tabu search heuristic for vehicle routing problems. *Computers & Operations Research*, 39(9):2033–2050, 2012.

[12] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[13] Richard Dawkins. *The selfish gene*. Number 199. Oxford university press, 1976.

[14] Humberto César Brandâo De Oliveira, José Lima Alexandrino, and Mariane Moreira De Souza. Memetic and genetic algorithms: A comparison among different approaches to solve vehicle routing problem with time windows. In *null*, page 55. IEEE, 2006.

[15] Guy Desaulniers, Jacques Desrosiers, and Marius M Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[16] W Dullaert. Impact of relative route length on the choice of time insertion criteria for insertion heuristics for the vehicle routing problem with time windows. In *Proc. Rome Jubilee 2000 Conf. Improving Knowledge Tools Transportation Logist*, pages 153–156, 2000.

[17] Fabrizio Ferrandi. Probabilistic (average-case) analysis. Technical report, Politecnico di Milano, 2014. Advanced Algorithms notes.

[18] Hermann Gehring and Jörg Homberger. A parallel hybrid evolutionary metaheuristic for the vehicle routing problem with time windows. In *Proceedings of EUROGEN99*, volume 2, pages 57–64. Springer Berlin, 1999.

[19] Mitsuo Gen and Runwei Cheng. *Genetic algorithms and engineering optimization*, volume 7. John Wiley & Sons, 2000.

[20] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The Vehicle Routing Problem: Latest Advances and New Challenges: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.

[21] Toshihide Ibaraki, Shinji Imahori, Koji Nonobe, Kensuke Sobue, Takeaki Uno, and Mutsunori Yagiura. An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, 156(11):2050–2069, 2008.

[22] George Ioannou, Manolis Kritikos, G Prastacos, et al. A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537, 2001.

[23] Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511, 2008.

[24] Dengying Jiang, Wenxia Jiang, and Zhangcan Huang. A hybrid algorithm for vehicle routing problem with time windows. In *Advances in Computation and Intelligence*, pages 198–205. Springer, 2008.

[25] Brian Kallehauge, Natashia Boland, and Oli BG Madsen. Path inequalities for the vehicle routing problem with time windows. *Networks*, 49(4):273–293, 2007.

[26] Leila Kallel and Marc Schoenauer. Alternative random initialization in genetic algorithms. In *Proceedings of the 7 th International Conference on Genetic Algorithms*, pages 268–275. Morgan Kaufmann, 1997.

[27] Borhan Kazimipour, Xiaodong Li, and AK Qin. A review of population initialization techniques for evolutionary algorithms. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 2585–2592. IEEE, 2014.

[28] Jan Karel Lenstra and AHG Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227, 1981.

[29] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[30] Weimin Ma, Yonghuang Hu, and Yang Zhou. An improved double-population genetic algorithm for vehicle routing problem with soft time windows. In *Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on*, pages 133–136. IEEE, 2010.

[31] Heikki Maaranen, Kaisa Miettinen, and Marko M Mäkelä. Quasi-random initial population for genetic algorithms. *Computers & Mathematics with Applications*, 47(12):1885–1895, 2004.

[32] Michael L Mauldin. Maintaining diversity in genetic search. In *AAAI*, pages 247–250, 1984.

[33] Pablo Moscato et al. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech concurrent computation program, C3P Report*, 826:1989, 1989.

[34] Yuichi Nagata. Efficient evolutionary algorithm for the vehicle routing problem with time windows: Edge assembly crossover for the vrptw. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1175–1182. IEEE, 2007.

[35] Yuichi Nagata and Olli Bräysy. Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. *Networks*, 54(4):205–215, 2009.

[36] Yuichi Nagata, Olli Bräysy, and Wout Dullaert. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers & Operations Research*, 37(4):724–737, 2010.

[37] Ibrahim H Osman and N Christofides. Simulated annealing and descent algorithms for capacitated clustering problem. *Imperial College, University of London, Research Report*, 1989.

[38] Jean-Yves Potvin and Jean-Marc Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, 1993.

[39] Christian Prins. A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers & Operations Research*, 31(12):1985–2002, 2004.

[40] Stefan Røpke. Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. *Presentation in Column Generation*, 2012.

[41] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4(1):285–305, 1985.

[42] Linus Schrage. Formulation and structure of more complex/realistic routing and scheduling problems. *Networks*, 11(2):229–232, 1981.

[43] Marius M Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research*, 35(2):254–265, 1987.

[44] Frank A Tillman and Thomas M Cain. An upperbound algorithm for the single and multiple terminal delivery problem. *Management Science*, 18(11):664–682, 1972.

[45] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*, volume 18. SIAM, 2014.

[46] Marta Vallejo, Patricia Vargas, David W Corne, et al. A fast approximative approach for the vehicle routing problem. In *Computational Intelligence (UKCI), 2012 12th UK Workshop on*, pages 1–8. IEEE, 2012.

[47] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. *Operations Research*, 60(3):611–624, 2012.

[48] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, 40(1):475–489, 2013.

[49] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, and Christian Prins. Time-window relaxations in vehicle routing heuristics. *Journal of Heuristics*, 21(3):329–358, 2013.

[50] Yi-Liang Xu, Meng-Hiot Lim, and Meng-Joo Er. Investigation on genetic representations for vehicle routing problem. In *Systems, Man and Cybernetics, 2005 IEEE International Conference on*, volume 4, pages 3083–3088. IEEE, 2005.